RedTitan User Guides

RedTitan Script Two

Copyright RedTitan Technology Ltd 2014 - 2018

RedTitan Script Two

EscapeE Software Development Kit scripting language

Published October 2018

All rights reserved. The information contained in this manual is confidential and is the property of RedTitan Limited. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of RedTitan Limited. It is made available on the understanding that it will not be disclosed without the written authority of RedTitan and that the information herein will not be used for any purpose other than instruction in the use of the RedTitan software.

In the interests of users RedTitan are constantly developing and improving their products and new versions of RedTitan Script Two are released from time to time. This manual may therefore differ in some ways from the version in use.

RedTitan and the RedTitan logo are registered trademarks of RedTitan Technology Ltd. Other trademarks are the properties of their respective owners.

While every precaution has been taken in the preparation of this document RedTitan assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall RedTitan be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

https://www.pclviewer.com

Table of Contents

Part I **About RS/2** **5**

RS/2 syntax 5

Getting started 8

Part II **RS/2 functions and procedures** **10**

Strings 11

Numbers 13

Values 15

Dialogs 16

Lists 17

Columns 18

Drawing 21

Tables 24

Format 27

Paper 31

Files and resources 33

Processing 36

Part III **Examples** **39**

Column example 39

InputBox example 41

Line example 42

Query example 43

Rotate example 44

Table example: Basic 45

Table example: Variable data 46

WriteLn example 47

Index **48**

Part I

About RS/2

About RS/2

RS/2 ("RedTitan Script Two") is part of the *EscapeE Software Development Kit*. It is the lightweight scripting language, based on Pascal, which provides RedTitan EscapeE with dynamic document features and extended field processing.

Three versions are available:

Version	Deploy	Usage
EVALUATE	EVALUATE.EEP must exist in the PLUGINS sub-directory of the ESCAPEE.EXE software folder.	Configured using EscapeE FIELD-ADVANCED. The evaluate plug-in will engage an RS/2 program (or a simple expression) to process a field.
RTRS2IN	RTRS2IN.DLL is co-located with ESCAPEE.EXE in the software folder.	EscapeE uses this filter to read and execute RS/2 programs (*.RS2) or the content of an RS/2 element in IDF.
RS2	RS2.EXE can be in any directory in the path.	RS2.EXE is a freestanding compiler and runtime system that can be used to execute RS/2 programs that do not require EscapeE capabilities. It is launched from the command line.

You can use RS/2 to dynamically create a new file by combining other resources, testing fields or processing disk files. An RS/2 program is a 'first class' EscapeE file – you can export the output to any supported format just like any other file.

EscapeE drawing capabilities can be accessed from an RS/2 program: see RS/2 syntax 5.

RS/2 syntax

The RS/2 language is a limited sub-set of Pascal.

- There are no type declaration blocks and types 7 are restricted to NUMBER, STRING, LIST and BOOLEAN.
- RS/2 requires the semi-colon in every position where Pascal permits.
- RS/2 has no runtime error messages. There is no exception handling.
- There are no objects or class syntax. Complex types are referenced using a typed handle, e.g. File handle.
- In IDF notation <RS2> tag may be used to introduce an RS/2 program.

The RS/2 source file is identified by the **preamble**

```
// REDTITAN RS2 CONTROL
```

Any other instance of // in a script indicates the start of a **comment** which RS/2 ignores. A comment is terminated at the end of the line of text.

Identifiers

An identifier type is declared by its first use;

Example assignment	Type
`A:=3;`	Number
`PI:=3.142;`	Number
`LX:=[];`	List (empty)
`S:='';`	String (empty)

An identifier may *not* be re-declared as a different type.

Statements

The following constructs are supported:

```
BEGIN ... END
IF ... THEN ... ELSE
CASE ... OF ...
REPEAT ... UNTIL ...
WHILE ... DO ...
FOR ... TO  ... DO ...
FUNCTION ...(...) BEGIN ... END
BREAK
EXIT
```

Operators

The following operators are available:

+	addition
-	subtraction
*	multiplication
/	division
¬	NOT
&	AND
!	OR
%	MOD

There is no operator order precedence, expression evaluation is left to right. Brackets may be be used specify an order.

- `writeln(3+4/5);` evaluates to `1.4`
- `writeln(3+(4/5));` evaluates to `3.8`

Runtime environment

RS/2 running in the freestanding RS/2 compiler has access to a console (WRITELN 37), a limited number of Windows® dialogs (SHOWMESSAGE 16, INPUTBOX 16, BROWSE 16) and access to the filestore (OPENFILE 33).

RS/2 running within *EscapeE* may also access a number of functions to draw in the EscapeE Window. This includes line drawing, text composition, display graphical resources, and field manipulation. If EscapeE is used to open an RS/2 file, the RS/2 program is executed for every page viewed or processed. EscapeE calls the RS/2 program to draw the selected page and passes the page number as a parameter.

Functions (private)

A user-defined function must be defined before first use. The formal parameter list may not contain VAR directives and may not be empty. The following function result types are permitted:

```
STRING
NUMBER
BOOLEAN
LIST
typed HANDLE
```

The private function returns a value by assigning a value to the RESULT identifier.

Notes on Types

NUMBER type

Numbers (Integer or Real) are held internally as floating point numbers.

STRING type

Strings are wide (16bit characters).

LIST type

An RS/2 list is a dynamic array of values, a type unique to RS/2. A list identifier may be instantiated using a bracketed array of constants:

```
LX:=['alpha',"beta",alpha,3.142];
```

The Pascal extended string syntax is permitted for an individual element but not mandated. As the RS/2 syntax is relaxed, comments 5 are not allowed in constant list definitions. Note that in this example the third constant element (alpha) is treated as if it was quoted (i.e. it is not a reference to an identifier 6). This format is intended as a way of storing a large number of constant elements in an easy interchange format (like a paragraph of text).
A number of functions are provided to manipulate lists,

```
LX:=['1.234',4,alpha,#13#10'new line',"element 4"];
writeln(list_numbers(lx,0)+list_numbers(lx,1));
writeln(list_strings(lx,3));
writeln(list_numbers(lx,4));
```

Elements are extracted from a list using the functions LIST_NUMBERS 17 or LIST_STRINGS 17. Where possible there is implicit type coercion between a string and a number when the list is first created. If a string cannot be interpreted as a number the value `-1` is returned.

Getting started

RS/2 scripts are text files that must start with the line

```
// REDTITAN RS2 CONTROL
```

The RS/2 program should signal the end of file condition to *EscapeE*. using a HALT 36 function with an error code greater than 0

```
// REDTITAN RS2 CONTROL
text(600,600,'This is page '+paramstr(1));
if paramstr(1)='5' then halt(1);
```

The example above will create a 5 page file when it is opened by EscapeE. See RS/2 procedures 10 for a list of the built-in standard functions and procedures.

An RS/2 file may be created with a simple text editor. The RS/2 file encoding format is UTF8. A plain text file is acceptable if the source text only contains characters in the ASCII character range $00 to $7F. If string constants contain characters outside this range or you wish to include other symbols then use a text editor that will display these characters (e.g. Microsoft® *Notepad*); select **Save As...** from the File menu and set the **Encoding** to **UTF-8**. A UTF-8 file will start with a sequence of special Byte Order Marks and UNICODE sequences will be encoded in UTF8 format. The RS/2 compiler can detect and repair some encoding problems automatically.

The Pound signs will be displayed correctly:

```
// REDTITAN RS2 CONTROL
flash('ff'); // Although these characters are ASCII $A3 (outside the
$00-$7F range)
halt(1);      // This file may be stored in plain text format
```

Greek letters, for example, will not be correct unless the source file has been stored in UTF-8 encoding

```
// REDTITAN RS2 CONTROL
flash('□XΦ'); // These characters are UNICODE $03A9 $03A7 $03A6
halt(1);      // This file should be stored in UTF-8 format
```

To learn more about UTF-8 see http://www.pcl.to/unicode/utf.xml

To automate the entry of unicode characters in simple editors like Notepad, download the RedTitan *Autoclick* utility – see http://www.pcl.to/unicode/

Links
RS/2 procedures 10
Examples 39

Part II

RS/2 functions and procedures

RS/2 functions and procedures

RS/2 functions and procedures are *case insensitive*. Click a link in the list of RS/2 standard functions and procedures below to see a full description.

AB
Box 22
Browse 16
Brush 21
Button 22

C
Char 12
Circle 22
CloseFile 34
Col_Blank 19
Col_Break 19
Col_Compose 19
Col_Count 20
Col_Create 18
Col_Display 19
Col_Font 18
Col_Free 19
Col_Height 20
Col_Leading 20
Col_Style 18
Col_Text 18
Colo[u]r 21
Copies 31
Copy 11
CreateFile 34
Curve 23
Custom 32

D
Dec 13
Delete 11

E
EOF 33
Even 13
Extent 29

FG
FieldValue 15
FileExists 34
Flash 16
Font 29
Format 27
FormatFloat 28
Frame 23

H
Halt 36

IJ
Inc 13
InputBox 16
IntToStr 14

KL
Length 17
Line 21
List_Add 17
List_Clear 17
List_IndexOf 17
List_Numbers 17
List_Strings 17
LowerCase 12

M
Max 14
Min 14

N
NewFile 33
Now 30

O
Odd 13
OpenFile 33
Ord 12
Orient 31

P
Paper 31
ParamCount 36
ParamStr 36
Pen 21
Pipe 37
Plex 31
PolyLine 23
Pos 11
Pred 15

Q
Query 16

R
Random 14
ReadCSV 33
ReadLine 33
Resource 34
ResourcePDF 35
Rotate 29

S
SetTrays 32
ShowMessage 16
Sleep 36
Split 12
StrToFloatDef 11
StrToIntDef 11
Succ 15

T
Tab 24
Tab_Align 24
Tab_Borders 26
Tab_Cell 24
Tab_Format 26
Tab_Home 25
Tab_NextRow 25
Tab_Pad 25
Tab_VAlign 25
Text 29
Trim 11

UV
UpperCase 12

WXYZ
WriteLine 34
WriteLn 37

The command key "**`markup:[`**" and "**`markup:]`**" is used to introduce variable data merge parameters. These parameters are name=value pairs and are added to paragraphs by the UBERED property editor or the CSV merge wizard. UBERED uses these sequences to dynamically create RS/2 scripts for variable data merge applications. These parameters are processed by UBERED in combination with a mark-up template (DEFAULT.MUP).

Strings

Copy(S:string; Index, Count: number): string;

Function

COPY returns a substring containing COUNT characters starting at S[INDEX]. If INDEX is larger than the length of S, then COPY returns an empty string or array.
If COUNT specifies more characters than are available, only the characters from S[INDEX] to the end of S are returned.

StrToIntDef(S: string; Default: number): number;

Function

STRTOINTDEF converts the string S (which represents a number in either decimal or hexadecimal notation) into a number.
If S does not represent a valid number, STRTOINTDEF returns the DEFAULT number.

Column example 39

See also STRTOFLOATDEF 11 and INTTOSTR 14 functions.

StrToFloatDef(S: string; Default: number): number;

Function

STRTOFLOATDEF converts the string S (representing a decimal number) to a floating-point number.
If S cannot be converted the DEFAULT is returned.

See also STRTOINTDEF 11 and INTTOSTR 14 functions.

Delete(var S: string; Index, Count:number);

Procedure

DELETE removes a substring, COUNT characters long, from string S starting with S[INDEX].
If INDEX is larger than the length of the string or less than 1, then no characters are deleted.
If COUNT specifies more characters than remain starting at the INDEX, then DELETE removes the rest of the string.
If COUNT is less than or equal to 0, no characters are deleted.

Pos(Substr: string; S: string): number;

Function

POS searches for a substring SUBSTR within a string S. POS is case-sensitive.
If SUBSTR is found then a number value that is the index of the first character of SUBSTR within S is returned.
If SUBSTR is *not* found then POS returns zero.

Trim(const S: string): string;

Function

TRIM removes whitespace – leading and trailing – from the string expression specified by S.

Table example: Variable data 46

LowerCase(const S: string): string;

Function

LOWERCASE returns string **S** converted to lower-case.

See also UPPERCASE 12 function, below.

UpperCase(const S: string): string;

Function

UPPERCASE returns string **S** converted to upper-case.

See also LOWERCASE 12 function, above.

Char(N:number):string;

Function

CHAR constructs a string consisting of a single wide character
$ffff>=N<=0

See also ORD 12 function, below.

Ord(const S: string): number;

Function

ORD returns the ordinal value of the first wide character of the string specified by **S**.
The result will be in the range **0..$ffff**

See also CHAR 12 function, above.

Split(Source,Separator:string;var LX:list);

Procedure

SPLIT procedure splits the **SOURCE** string into a number of elements delimited by the **SEPARATOR** string. The elements are returned in the list specified by the **LX** parameter. The previous contents of the list are lost.
If the **SEPARATOR** does not exist in the **SOURCE** string then the list will contain a single entry consisting of the entire **SOURCE** string.

```
split('alpha/beta/gamma','/',LX);
```

See also READCSV 33 procedure.

Numbers

Even(N:number):boolean;

Function

EVEN function returns **TRUE** if number **N** is not odd.

See also ODD 13 function, below.

Odd(N:number):boolean;

Function

ODD function returns **TRUE** if number **N** is odd.

See also EVEN 13 function, above.

Inc(var X [; N: number]);

Procedure

X is a number-type variable.
N is an optional number-type expression.

If **N** is specified then **X** increments by **N**; that is, **INC(X, N)** corresponds to the statement **X := X + N**.
If **N** is *not* specified then **X** increments by **1**; that is, **INC(X)** corresponds to the statement **X := X + 1**.
Note that **INC** generates optimized code and is especially useful in tight loops.

See also DEC 13 procedure, below.

Dec(var X[; N: number]);

Procedure

X is a number-type variable.
N is a number-type expression.

If **N** is specified then **X** decrements by **N**; that is, **DEC(X, N)** corresponds to the statement **X := X - N**.
If **N** is *not* specified then **X** decrements by **1**; that is, **DEC(X)** corresponds to the statement **X := X - 1**.
Note that **DEC** generates optimized code and is especially useful in tight loops.

See also INC 13 procedure, above.

Min(A,B: number): number;

Function

MIN returns the lower of the two values **A** and **B**; negative values are smaller than zero. Thus:

When	MIN returns	Example A	Example B	Example MIN
A>B	B	2	1	1
A<B	A	-2	-1	-2
A=B	B	2	2	2

See also MAX 14 function, below.

Max(A,B: number): number;

Function

MAX returns the higher of the two values **A** and **B**; negative values are smaller than zero. Thus:

When	MAX returns	Example A	Example B	Example MAX
A>B	A	2	1	2
A<B	B	-2	-1	-1
A=B	B	2	2	2

See also Min 14 function, above.

Random [Range: number]:number;

Function

RANDOM function returns a random number within the range:
`0 <= X < RANGE.`

IntToStr(N:number):string;

Function

INTTOSTRING converts number **N** to string representation.

Rotate example 44, Column example 39

See also STRINGTOINTDEF 11 and STRTOFLOATDEF 11 functions.

Values

FieldValue(Name:string):string;

Function

In EscapeE, **FIELDVALUE** will return the value of a mark-up or "composite" field.

```
s:=FieldValue('{field1}');
```

In the Windows RS2 JIT compiler, **FIELDVALUE** will return an empty string. It can be used to detect the supporting platform.

```
if fieldvalue('{_Iname}')='' then
begin
  showmessage('Please open with RedTitan EscapeE',3000);
  halt(1);
end;
```

Pred(I:number):number;

Function

The **PRED** function is equivalent to the assignment `I:=I-1;`

See also SUCC 15 function, below.

Table example: Variable data 46.

Succ(I:number):number;

Function

The **SUCC** function is equivalent to the assignment `I:=I+1;`

See also PRED 15 function, above.

Dialogs

InputBox(const ACaption, APrompt, ADefault: string): string;

Function

A modal dialog containing an "editable box" is displayed. **InputBox** function returns the string entered into this box by the user or, if the box is not edited, a default string.
ACaption specifies the text to be used in the title bar of the dialog.
APrompt specifies the text shown above the editable box.
ADefault supplies the text to be used if the editable box is not edited.

InputBox example 41

Flash(S:string);

Procedure

FLASH displays non-modal text alert window until another modal dialog or a new top window is selected.
String **s** contains the text to be displayed.

Query example 43

Query(Caption:string;[Timeout:number;Default:boolean]):boolean;

Function

QUERY function displays a dialog requesting a YES or NO response.
TIMEOUT (optional) parameter specifies the number of milliseconds to wait for user response.
DEFAULT (optional) parameter specifies whether the result is **TRUE** or **FALSE** if the dialog times out.

Query example 43

ShowMessage(S:string;[Timeout:number]);

Procedure

SHOWMESSAGE displays a modal dialog alert message.
The optional parameter **TIMEOUT** specifies the number of milliseconds to wait for user response.

Browse[(Caption[[,InitialDir],Filters]:string)]:string;

Function

BROWSE function returns the string resulting from a "Browse for a file" dialog.
CAPTION default value is **'Select file'**.
INITIALDIR default value is an empty string **''**.
FILTERS default value is **'*.*|*.*'**.

```
j:=browse(
   'Select',
   'c:\eeplugin',
   'Zip (*.zip)|*.zip|Async (*.abb)|*.abb|Async (*.asy)|*.asy'
   );
```

Tip: your RS/2 script should check for an empty string return: this indicates the dialog has been cancelled.

Table example: Variable data 46

Lists

Length(LS:string|list): number;

Function

If **LS** is a *string* expression, then **LENGTH** returns the number of *characters* in the string. If **LS** is a *list*, then **LENGTH** returns the number of *elements* in the list.

Table example: Variable data 46

See also LIST_STRINGS 17 function.

List_Strings(L:list;I:number):string;

Function

LIST_STRINGS function returns the element of list **L** specified by index number **I** as a string.

Table example: Variable data 46

See also LENGTH 17, LIST_NUMBERS 17 functions.

List_Add(L:list;S:string):number;

Function

LIST_ADD adds a string element **S** to the list **L** then returns the element index number.

List_Clear(L:list);

Procedure

LIST_CLEAR removes all elements in list **L**.

List_Numbers(L:list;I:number):number

Function

LIST_NUMBERS returns element **I** of list **L** as a number.
If the list element cannot be interpreted as a number then **LIST_NUMBERS** returns **-1**.

See also LENGTH 17, LIST_STRINGS 17 functions.

List_IndexOf(L:list;SearchFor:string;[CaseBlind:boolean=true;[StartIndex:number=0]]):number;

Function

LIST_INDEXOF returns the index of a specified string **SEARCHFOR** in list **L**. If the specified string is not found then **LIST_INDEXOF** returns **-1**. The function defaults to case-blind string matching and searches from the first element of the list.

```
l:=['alpha','Beta','gamma'];
writeln(list_indexof(l,'beta')); // returns 1
writeln(list_indexof(l,'beta',false)); // returns -1
```

See also LIST_STRINGS 17, LIST_NUMBERS 17 functions.

Columns

Col_Create:cHandle;

Function

EscapeE only

COL_CREATE function creates a column-formatting resource. All "text in column" formatting procedures require a column object handle – CHANDLE. There must be a corresponding COL_FREE 19 call for each COL_CREATE function.

```
cx:=col_create;
for i:=1 to 30 do col_text(cx,'hello world ');
col_compose(cx,4200,'rj');
col_display(cx,300,2600);
col_free(cx);
```

Column example 39

Col_Text(Cx:cHandle;S:string);

Procedure

EscapeE only

COL_TEXT adds text S to a "text in column" formatting object referenced by column object handle CX. No text is drawn by this operation. Any amount of text may be added before the object is composed and displayed.

Use COL_FONT 18 and COL_STYLE 18 procedures to control the font face and style used by COL_TEXT.

Column example 39

Col_Font(Cx:cHandle;Pt:number;FontName:string);

Procedure

EscapeE only

COL_FONT selects the font to be used for the COL_TEXT 18 procedure.

```
col_Font(cx,12,'Tahoma');
```

Column example 39

Col_Style(Cx:cHandle;Style:string);

Procedure

EscapeE only

COL_STYLE selects the font style to be used for the COL_TEXT 18 procedure.

STYLE options are BOLD and ITALIC; an empty string selects "NORMAL" style.

```
col_Style(cx,'Bold');
```

Column example 39

Col_Compose(Cx:cHandle;Width:number;Format:string);

Procedure

EscapeE only

COL_COMPOSE prepares the "text in column" formatting object for display. Specify a column **WIDTH** (in pixels at 600dpi) and a column **FORMAT** option – '**LJ**', '**CJ**', '**RJ**' or '**FILL**'. The column's height and line count properties are calculated for the chosen option; these values may be recovered using the COL_HEIGHT 20 and COL_COUNT 20 functions. The stored text is not destroyed by this procedure and it may be repeated for the same text with different parameters.

Column example 39

See also COL_LEADING 20 procedure.

Col_Display(Cx:cHandle;X,Y:number);

Procedure

EscapeE only

COL_DISPLAY draws text, previously formatted using the COL_COMPOSE 19 procedure, at the specified position **X,Y** on the page. The stored text is not destroyed by this procedure and it may be repeated for the same text with different parameters.

Column example 39

Col_Free(Cx:cHandle);

Procedure

EscapeE only

COL_FREE recovers column formatting resources. See COL_CREATE 18 function.

Column example 39

Col_Break(Cx:cHandle);

Procedure

EscapeE only

COL_BREAK inserts a line break in a column formatting object.

Column example 39

Col_Blank(Cx:cHandle;Lines:number);

Procedure

EscapeE only

COL_BLANK inserts a number of blank **LINES** in a column formatting object.

Column example 39

Col_Height(Cx:cHandle):number;

Function

EscapeE only

COL_HEIGHT returns the composed height of a column.

Column example 39

See also COL_COUNT 20 function and COL_COMPOSE 19, COL_LEADING 20 procedures.

Col_Count(Cx:cHandle):number;

Function

EscapeE only

COL_COUNT returns the number of lines in a composed column (see COL_COMPOSE 19 procedure).

Column example 39

Col_Leading(Cx:cHandle;L:number);

Procedure

EscapeE only

COL_LEADING allows the inter-line gap to be adjusted before a column is displayed (see COL_DISPLAY 19 procedure).

Column example 39

Drawing

Line(X1,Y1,X2,Y2:number);

Procedure

EscapeE only.

LINE procedure draws a line from **X1,Y1** to **X2,Y2** using the the current PEN 21 and COLOR 21, see below.

- Line example 42

Color(Red,Green,Blue: number);
Colour 21(Red,Green,Blue: number);

Procedure

EscapeE only.

COLOR procedure (*or* the alternative spelling **COLOUR**) specifies a color for TEXT 29 and LINE 21 drawing.
RED, **GREEN** and **BLUE** may take values **>=0**, **<=255** in decimal or **>=$00**, **<=$FF** in hexadecimal.

- **COLOR(255,128,0)** would be orange. (Equivalent to **COLOR($FF,$80,$00)** in Hex.)
- Rotate example 44, Line example 42.

See also BRUSH 21 procedure below.

Brush(Red,Green,Blue: number);

Procedure

EscapeE only.

BRUSH specifies the color for filled regions; see BOX 22 and BUTTON 22 procedures below.
RED, **GREEN** and **BLUE** may take values **>=0**, **<=255** in decimal or **>=$00**, **<=$FF** in hexadecimal.

- **BRUSH(0,255,255)**

would be cyan. (Equivalent to **BRUSH($00,$FF,$FF)** in Hex.)

See also COLOR 21 and FRAME 18 procedures.

Pen(Width:number);

Procedure

EscapeE only.

PEN specifies the pen's **WIDTH** in pixels (at 600dpi) for LINE 21 drawing.

- Table example: Variable data 46

Box(X,Y,Dx,Dy:number);
Box 22 **(TH:tableHandle);**

Procedures

EscapeE only.

BOX fills an area **DX** wide by **DY** high at position **X,Y** (in pixels at 600dpi) using the color specified by BRUSH 21.
The **BOX** function optionally takes a tableHandle **TH** (see TAB 24 function).

See also the FRAME 23 and BUTTON 22 functions.

Button(X,Y,Dx,Dy:number);
Button 22 **(TH:tableHandle);**

Procedures

EscapeE only.

BUTTON draws a filled button shape (lozenge) in the selected BRUSH 21 color.
The **BUTTON** function optionally takes a tableHandle **TH** (see TAB 24 function).

See also the FRAME 23 and BOX 22 functions.

Circle(X,Y,R[,Section]:number);

Procedure

EscapeE only.

CIRCLE draws a complete circle or a **SECTION** of a circle, radius **R** centered at position **X,Y,** (measured in pixels at 600dpi) using the current PEN 21 and COLOR 21.
If **SECTION** parameter is omitted a complete circle is drawn.

section=0 draws a complete circle (default)
section=1 draws the top right quadrant
section=2 draws the bottom right quadrant
section=3 draws the bottom left quadrant
section=4 draws the top left quadrant

Rotate example 44, Line example 42.

Curve(L:list);

Procedure

EscapeE only.

CURVE displays a Bezier curve.
L is a list of point-coordinate pairs, consisting of a start point pair (x,y) followed groups of three coordinate pairs. (Specified in pixels at 600dpi.)

```
L:=
[750, 1350,
150,750,   750,150,   1350,750,
1950,1350, 2550,1350, 3150,750,
...
```

See rs2 demo files and choose **poly.rs2**

PolyLine(L:list);

Procedure

EscapeE only.

Displays a **POLYLINE** linking all points specified by lists of coordinates in list **L**.

See rs2 demo files and choose **poly.rs2**

Frame(X,Y,Dx,Dy:number);
Frame 23 **(TH:tHandle);**

Procedure

EscapeE only.

FRAME draws a rectangle **DX** wide by **DY** high at **X,Y** (measured in pixels at 600dpi) using the selected PEN 21 and COLOR 21.
An overload of this function takes a **T**able **H**andle (see TAB 24 function): a filled rectangle is drawn using the BOX 22 function.

Tables

Tab(Xpos,Ypos:number;Htabs:list;Vspace:number):tableHandle;

Function

TAB defines a table.

XPOS,YPOS specify the position of the top-left corner of the table.
HTABS is a comma-separated list of default column widths (in pixels at 600dpi). There must be at least one value in the list. If there are fewer widths specified than the data requires, extra columns are added at the right of the table: the last value in the list is used to supply their default width. If there are more widths specified than the data requires, the extra values are ignored: no spurious columns are generated.
VSPACE specifies the default row height.

A number of related functions may be used to display table cells containing text with simple formatting: see TAB_ALIGN 24, TAB_BORDERS 26, TAB_CELL 24, TAB_PAD 25, TAB_VALIGN 25 procedures and TAB_FORMAT 26, TAB_HOME 25, TAB_NEXTROW 25 functions.

Table example: Basic 45, Table example: Variable data 46.

Tab_Cell(TH:tableHandle;S:string);

Procedure

TAB_CELL displays text **S** in a cell of the table specified by **TH**.

Table example: Basic 45, Table example: Variable data 46.

See TAB 24 function, above.

Tab_Align(TH:tableHandle;AlOpt:string);

Procedure

TAB_ALIGN specifies the *horizontal* text alignment.

ALOPT may be:

L for Left,
C for Center,
R for Right *or*
N for None. This option suppresses any composition in the FORMAT 27 and TAB_FORMAT 26 functions.

Table example: Basic 45, Table example: Variable data 46.

See also TAB_VALIGN 25 procedure, below.

Tab_VAlign(TH:tableHandle;AlOpt:string);

Procedure

TABVALIGN specifies the *vertical* text position.

ALOPT may be:

T for Top,
M for Middle *or*
B for Bottom

See also TABALIGN 24 procedure, above.

Tab_Pad(TH:tableHandle;Padding:number);

Procedure

TAB_PAD specifies the offset from the edge of cell to the text position, in pixels at 600dpi.

Table example: Basic 45, Table example: Variable data 46.

Tab_NextRow(TH:tableHandle[;Rulings:boolean=true]):number;

Function

TAB_NEXTROW is used to start a new row of a table: it returns the vertical position of the top of the next row down (in pixels at 600dpi). Row height is determined by the maximum of the vertical spacing specified by the **T**able **H**andle and the tallest text composed on the row.

RULINGS (optional) set to **TRUE** to draw lines at cell boundaries.

Table example: Basic 45, Table example: Variable data 46.

See also PEN 21 and TAB_PAD 25 procedures.

Tab_Home(TH:tableHandle):number;

Function

TAB_HOME returns to the first column cell position without advancing the vertical position. This function my be used to add additional text in each cell (in a different position) or add borders in particular positions.
Cell drawing may be repeated using the cell height determined by the maximum of the vertical spacing specified by the **T**able **H**andle and the tallest text composed on the row.

Tab_Borders(TH:tableHandle;Borders:string);

Procedure

TAB_BORDERS adds borders to a cell.

Borders options are a combination of:

T for TOP
L for LEFT
B for BOTTOM
R for RIGHT

Tab_Format(TH:tableHandle;FormattedText:list):number;

Function

TAB_FORMAT adds formatted text to a cell. See FORMAT 27 parameters for **FORMATTEDTEXT** list options.

Format

Format(AttributeList,FormattedText:list):number;

Function

FORMAT and **TAB_FORMAT** are analogues of the IDF Rich Text container used in the dynamic composition number of text paragraphs.
ATTRIBUTELIST specifies the overall text attributes as an ordered list of up to 13 properties as follows:

Left, Top, Width, Height, Padding, BorderWidth, Rotate, LineJoin, LineEnd, BorderStyle, Borders, BorderColor, BGColor

```
[600, 600, 900, 1050, 0, 0, 0, "round", "round", "solid", "T,L", "Black", "#408000"]
```

FORMATTEDTEXT list consists of "commandkey:value" pairs expressed as strings.

```
["text:hello","break:","text:world"]
```

Command key	Description	Example
text:	Sequence	text:hello world
font:	Font face name	font:Arial
size:	Point size	size:12
break:	Argument ignored. Insert line break	break:
blank:	Argument ignored. Insert blank line	blank:
align:	Set text alignment. LJ, RJ, CJ, or FILL	align:lj
bold:	Engage bold font	bold:
italic:	Engage italic font	italic:
underline:	Engage text	underline:
color:	Engage specified color	color:#0000ff

FormatFloat(S:string;N:number):string;

Function

FORMATFLOAT uses the string **S** to specify how the floating-point number **N** should be displayed. **S** may take these values:

0	Forces digit display or 0
#	Optional digit display
,	Forces display of thousands
.	Forces display of decimals
E+	Forces signed exponent display
E-	Optional sign exponent display
;	Separator of +ve and -ve and zero values

This table shows the effect of different strings on the way numbers are displayed.

string	N=1234	N=-1234	N=0.5	N=0
'0'	1234	-1234	1	0
'0.00'	1234.00	-1234.00	0.50	0.00
'#.##'	1234	-1234	.5	
'#,##0.00'	1,234.00	-1,234.00	0.50	0.00
'#,##0.00;(#,##0.00)'	1,234.00	(1,234.00)	0.50	0.00
'#,##0.00;;Zero'	1,234.00	-1,234.00	0.50	Zero
'0.000E+00'	1.234E+03	-1.234E+03	5.000E-01	0.000E+00
'#.###E-0'	1.234E3	-1.234E3	5E-1	

Font(PointSize:number;FontName,Style:string);

Procedure

EscapeE only

FONT engages the specified font.

- **Font(12,'Arial','bold');**
- Line example 42, Table example: Basic 45.

See also TEXT 29 and EXTENT 29 procedures, below.

Text(X,Y:number;S:string)

Procedure

EscapeE only

TEXT draws text **S** on page at **X,Y** (measured in pixels at 600dpi).

- Line example 42, Rotate example 44, Table example: Variable data 46.

See also FONT 29 procedure, above.

Rotate(Angle: number);

Procedure

EscapeE only.

ROTATE specifies the **ANGLE** in degrees counter-clockwise for TEXT 29 display.

- Rotate example 44, Line example 42.

Extent(S:string;var Ascent,Descent,Width:number);

Procedure

EscapeE only

EXTENT procedure returns text metrics for the string **S** in pixels at 600dpi.

ASCENT variable gives the font maximum height above the baseline.
DESCENT variable gives the text maximum depth below the baseline.
WIDTH variable gives the overall length of the string.

Now(Format:string):string;

Function

NOW specifies current date and time in one of these **Format** options:

y	Year; last 2 digits
yy	Year; last 2 digits
yyyy	Year as 4 digits
m	Month number; no-leading 0
mm	Month number as 2 digits
mmm	Month using ShortMonthNames (Jan)
mmmm	Month using LongMonthNames (January)
d	Day number; no-leading 0
dd	Day number as 2 digits
ddd	Day using ShortDayNames (Sun)
dddd	Day using LongDayNames (Sunday)
ddddd	Day in ShortDateFormat
dddddd	Day in LongDateFormat
c	Use ShortDateFormat + LongTimeFormat
h	Hour number; no-leading 0
hh	Hour number as 2 digits
n	Minute number; no-leading 0
nn	Minute number as 2 digits
s	Second number; no-leading 0
ss	Second number as 2 digits
z	Millisecond number; no-leading 0s
zzz	Millisecond number as 3 digits
t	Use ShortTimeFormat
tt	Use LongTimeFormat
am/pm	Use after h : gives 12 hours + am/pm
a/p	Use after h : gives 12 hours + a/p
ampm	As a/p but TimeAMString,TimePMString
/	Substituted by DateSeparator value
:	Substituted by TimeSeparator value

```
NOW('ddd mmm yyyy hh:mm:ss.zzz');
```

Column example 39

Paper

Orient(Direction:string);

Procedure

EscapeE only

ORIENT specifies the orientation of display. **DIRECTION** may take one of these values:
'P' = portrait (default)
'L' = landscape
'I' = inverse (portrait rotated 180degrees)
'J' = journal (landscape rotated 180degrees)

- Table example: Variable data 46

See also PAPER 31 and CUSTOM 32 procedures, below

Plex(Option:string);

Procedure

EscapeE only.

PLEX specifies whether one or both sides of the paper may be used. In the case of **DUPLEX**, edge-binding may be specified.

[SIMPLEX|DUPLEX][LONG|SHORT]

- **Plex('DUPLEX LONG');**
- Line example 42

See also ORIENT 31 procedure, above.

Copies(N:number);

Procedure

EscapeE only.

COPIES specifies the number, **N**, of copies required.

Paper(Option:string;Force:boolean);

Procedure

EscapeE only.

Set paper type.

- **Paper('A4',false);**
- Table example: Variable data 46

See also CUSTOM 32 procedure, below.

Custom(W,H:number);

Procedure

EscapeE only.

CUSTOM sets the width **W** and height **H** (in pixels at 600dpi) for custom paper.

See also PAPER 31 procedure, above.

SetTrays(T,B:number);

Procedure

EscapeE only.

Set input paper tray **T** and output bin **B**.

See also PAPER 31 procedure, above.

Files and resources

NewFile(Name:string);

Procedure

EscapeE only.

NEWFILE starts a new output file for this page. (**NAME** is currently ignored)

OpenFile(Name:string):fileHandle;

Function

OPENFILE returns a handle to a text file for reading. Use the handle in READLINE 33, READCSV 33 and EOF 33 functions.

```
h:=openfile('test.txt');
while not eof(f) do
  begin s:=readline(h); writeln(s); end;
```

Table example: Variable data 46

ReadLine(var Handle:fileHandle):string;

Function

READLINE returns the next string read from the file specified by the **HANDLE** identifier (created by OPENFILE 33).

See also READCSV 33 below.

ReadCSV(var Handle:fileHandle;var Lx:list;
[Separator:string[1]]);

Procedure

READCSV returns the next CSV record from the file specified by the **HANDLE** identifier (created by OPENFILE 33) in the list specified by **LX**.
If **SEPARATOR** is not specified then a comma is used to separate the elements. Each element will contain a field from the CSV record.

A multi-line field is normalized to a string containing a number of lines delimited by 'Line Feed' (#10) characters, i.e. 'CR' (#13) is not stored. Use the SPLIT 12 function to decompose multi-line fields.

Table example: Variable data 46

EOF(var Handle:fileHandle):boolean;

Function

EOF tests if any more data can be read from a file previously opened using OPENFILE 33. If **EOF** fails, the **HANDLE** is no longer valid, and the file is closed.

Table example: Variable data 46

CreateFile(FileName:string):fileHandle;

Function

CREATEFILE creates a file named **FILENAME** and returns a handle that can be used for WRITELINE 34 procedure and CLOSEFILE 34 function.

- Tip: "**CREATEFILE WRITELINE WRITELINE CLOSEFILE**" summarizes the file create process.

WriteLine(var Handle:number;S:string);

Procedure

WRITELINE writes a string **S** to a file previously opened with CREATEFILE 34 function (see above).

CloseFile(Handle:number):boolean;

Function

CLOSEFILE closes a file previously created with CREATEFILE 34 function.

FileExists(FN:string):boolean;

Function

FILEEXISTS returns **TRUE** if the file – specified by the file named by **FN** string – is extant.

- Table example: Variable data 46

Resource(FN:string; [X,Y:number;[Scale:number;[Page:number; [Transparent:boolean[,ClipLeft,ClipTop,ClipRight,ClipBottom:number]]]]]):boolean;

Function

EscapeE only

RESOURCE function adds the designated resource to the EscapeE page. If the resource cannot be found, or the requested page does not exist in the file, the function returns **FALSE**.

FN must be the name of a file in a format that EscapeE can view (PCL, PDF, TIFF, RS2 etc.) and is *mandatory*.
X optional number of pixels at 600dpi, default **0** (left-hand side of page).
Y optional number of pixels at 600dpi, default **0** (top of page).
SCALE optional number, default **1** (i.e. unscaled).
PAGE optional number, default **1**.
TRANSPARENT optional boolean, default **TRUE**.
CLIPLEFT, **CLIPTOP**, **CLIPRIGHT** and **CLIPBOTTOM** are optional numbers (default **0**) specifying a clip region.

- To draw a graphic at the top left-hand corner of the page:

```
resource('TEST.PNG');
```

ResourcePDF(FN:string; [X,Y:number;[Scale:number;[Page:number; [Transparent:boolean[,ClipLeft,ClipTop,ClipRight,ClipBottom:number]]]]]):boolean;

Function

EscapeE only

RESOURCEPDF function is used to engage special processing for PDF export only. The **RESOURCEPDF** File Name is only stored once in the output file and the content reused by each page as an embedded resource. If the resource cannot be found, or the requested page does not exist in the file, the function returns **FALSE**.

FN this string expression specifies the resource's File Name and is *mandatory*.
X optional number of pixels at 600dpi, default **0** (left-hand side of page).
Y optional number of pixels at 600dpi, default **0** (top of page).
SCALE optional number, default **1** (i.e. unscaled).
PAGE optional number, default **1**.
TRANSPARENT optional boolean, default **TRUE**.
CLIPLEFT, **CLIPTOP**, **CLIPRIGHT** and **CLIPBOTTOM** are optional numbers (default **0**) specifying a clip region.

In this example, the 50-page PDF output file is only a little larger than the 1-page source:

```
// REDTITAN RS2 CONTROL

resourcePDF('overlay.pdf');
text(100,100,'Page '+paramstr(1));

if paramstr(1)='50' then halt(1);
```

See also RESOURCE 34, above.

Processing

ParamCount:number;

Function

PARAMCOUNT returns the number of parameters passed to the program on the command line.
See also PARAMSTR 36 below.

ParamStr(Index: number): string;

Function

PARAMSTR returns the parameter from the command line that corresponds to **INDEX**, *or* if **INDEX** is greater than PARAMCOUNT 36, then **PARAMSTR** returns an empty string.
For example, an **INDEX** value of 2 returns the second command-line parameter.

In EscapeE the **INDEX** values 0 and 1 have special meanings:
PARAMSTR(0) returns the path and file name of the RS/2 program e.g. "Table.RS2".
PARAMSTR(1) supplies the page number.
In EscapeE (v8.85+) extended command line filename syntax, the filename parameter may be followed by a number of other parameters. Each parameter is separated from the next by a comma;
PARAMSTR(n) for **n>1** to **n<=PARAMCOUNT** returns the **n**th parameter on the command line.

- This command line contains the file name parameter "Table.RS2" followed by the name parameter of the data file used to fill the table "Sales.csv". (**/PDF** and **/X** are EscapeE switches – see EscapeE Help.)
  ...ESCAPEE Table.RS2,Sales.csv /pdf /x
  then **PARAMSTR(2)** would return "Sales.csv".
- Column example 39, Table example: Variable data 46.

Halt(N:number);

Procedure

In Windows RS2 JIT compiler, **HALT** terminates program execution and returns the exit code **N**.

In EscapeE, **HALT(1)** is used to signal 'end of file'.

- Column example 39, Rotate example 44, Line example 42, Table example: Basic 45, Table example: Variable data 46.

See also EOF 33 and CLOSEFILE 34 functions.

Sleep(Millisecs:number);

Procedure

SLEEP suspends processing for specified period.

Pipe(PipeName,Query:string):string;

Function

The RS/2 function **PIPE** provides inter-process communication on a local computer to another program that implements the pipe server named **PIPENAME**. For example, to send database queries to Microsoft® *SQL Server*®.
Strings of up to 8000 bytes may be sent and received.

See also NAMESERVER.ZIP in the rs2 demo files.

WriteLn(P1 [, P2, ...,Pn]);

Procedure

WriteLn writes lines of text to the *console* window.

In EscapeE this is the LOG window.

(To write to a *file* see WRITELINE 34 procedure.)

WriteLn example 47

Part III

Examples

Examples

This section contains worked examples.

To try an example

1. Sweep out the text in a blue box then copy and paste it into a text editor such as Microsoft® *Notepad*.
2. Save the file in plain TXT format but change the file extension to "RS2".
3. Open the RS2 file in *EscapeE*.

- Column 39
- InputBox 41
- Line 42
- Query 43
- Rotate 44
- Table: Basic 45
- Table: Variable data 46
- WriteLn 47

Column example

```
Col_Create:cHandle;
Col_Free(Cx:cHandle);
```

There are four columns of text shown on each of nine pages in this example:

Each of the four columns is composed with a different alignment:

Column1 FILL
Column2 CJ
Column3 LJ
Column4 RJ

The time at which each page was created is shown in Column4 using the NOW function.

(Text to be printed on the page shown in black.)

```
// REDTITAN RS2 CONTROL
PAGENUMBER:= strtointdef(paramstr(1),0);
if PAGENUMBER>10 then halt(1);

CX1:= col_create;  //COLUMN1
  col_font(CX1,14,'Tahoma');
  WORDS1:= 'Column1: use the COL_CREATE function to set up a column then
COL_TEXT procedure to enter the text. Specify the width of the column and
the alignment of the text within the column using COL_COMPOSE procedure.
COL_DISPLAY procedure places the composed column on the page then
COL_FREE procedure releases the handle set up by   COL_CREATE.';
  col_text(CX1,WORDS1);
  col_compose(CX1,1800,'fill'); col_display(CX1,600,600);
  H1:= col_height(CX1);
col_free(CX1);

CX2:= col_create;  //COLUMN2
  col_font(CX2,14,'Tahoma');  col_text(CX2,'Column2: use the COL_FONT
procedure to change the font ');
  col_font(CX2,18,'Tahoma');  col_text(CX2,'size ');
  col_font(CX2,14,'Tahoma');  col_text(CX2,'and ');
  col_font(CX2,14,'Times New Roman');  col_text(CX2,'typeface. ');
  col_font(CX2,14,'Tahoma');  col_text(CX2,'The COL_STYLE procedure is
often a more convenient way of making spot changes to font ');
  col_style(CX2,'italic');  col_text(CX2,'style ');
  col_font(CX2,14,'Tahoma');  col_text(CX2,'and ');
  col_style(CX2,'bold');  col_text(CX2,'weight.');
  col_font(CX2,14,'Tahoma');
  col_break(CX2);
  col_text(CX2,'COL_BREAK procedure forces a line-break in a column. A
COL_BLANK procedure here');
  col_blank(CX2,2);
  col_text(CX2,'inserts a line-break and two blank lines.');
  col_compose(CX2,1800,'cj'); col_display(CX2,2600,600);
  LINES2:= inttostr(col_count(CX2));
col_free(CX2);

CX3:= col_create;  //COLUMN3
  col_Font(CX3,14,'Tahoma');
  WORDS3:= 'Column3: the COL_HEIGHT function has been used here to make a
gap of 1 inch between this column and column1. The COL_COUNT function
returns the number of lines used to compose a column: there are '+LINES2
+' lines in column2.';
  col_text(CX3,WORDS3);
  col_compose(CX3,1800,'lj'); col_display(CX3,600,1200+H1);
col_free(CX3);

CX4:= col_create;  //COLUMN4
  col_Font(CX4,14,'Tahoma');
  WORDS4:= 'Column4: the COL_LEADING procedure inserts gaps between the
lines of text in a column. It is now '+now ('hh:nn:ss')+' on '+
now('ddddd')+' ("ShortDateFormat").';
  col_text(CX4,WORDS4); col_compose(CX4,1800,'rj');
  col_leading(CX4,150); col_display(CX4,2600,3600);
col_free(CX4);

if PAGENUMBER=9 then halt(1);
```

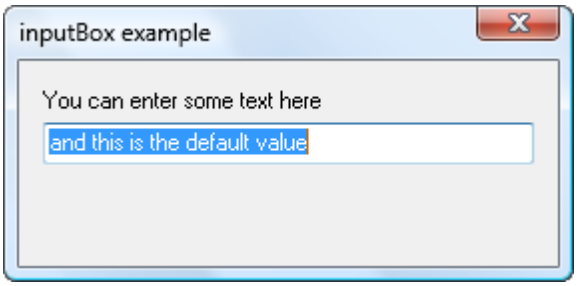
InputBox example

`InputBox(const ACaption, APrompt, ADefault: string): string;`

This RS/2 script:

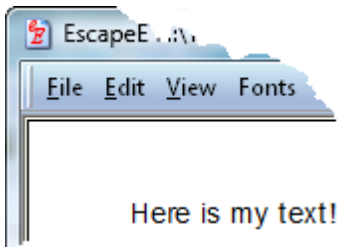
```
// REDTITAN RS2 CONTROL
s:=inputbox('inputBox example','You can enter some text here','and this is the default value');
text(300,300,s);
```

displays this modal dialog:



Try typing Here is my text! in the editable box to overwrite the **`ADefault`** string **`and this is the default value`**.

The *EscapeE* Applications Programming Interface will display:



Line example

Line(X1,Y1,X2,Y2:number);

Print this on a sheet of A4 paper then fold along the lines to make a paper dart.

```
// REDTITAN RS2 CONTROL

Plex('DUPLEX LONG');
Font(10,'Arial','');
PAGENUM:=strtointdef(paramstr(1),0);

case PAGENUM of
  1:begin
    line(2478,0,2478,7014);
    rotate(270);
    text(2600,2000,'1: fold paper in half along this line so that the
red circle is visible');
    color(120,120,220);
    line(0,1239,1239,1239); //line3 ext
    color(120,220,120);
    line(3717,1239,4956,1239); //line5 ext
  end;

  2:begin
    color(120,170,220);
    line(2478,0,4956,2478);
    rotate(315);
    text(3500,950,'2: fold this corner down');

    color(120,120,220);
    line(3717,1239,3717,7014);
    rotate(270);
    text(3750,3800,'3: fold edge down');

    color(120,220,170);
    line(2478,0,0,2478);
    rotate(45);
    text(1000,1350,'4: fold this corner down');

    color(120,220,120);
    line(1239,1239,1239,7014);
    rotate(90);
    text(1200,4250,'5: fold edge down');

    color(255,0,0);
    rotate(0);
    text(2000,3000,'6: hold here to launch');
    circle(2478,3000,500);
  end;

else halt(1);

end;
```

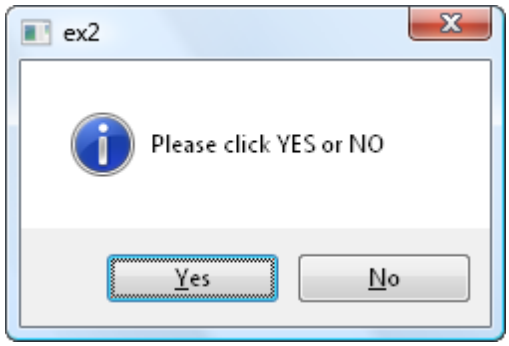
Query example

Query(Caption:string;[Timeout:number;Default:boolean]);

Running this RS/2 script:

```
// REDTITAN RS2 CONTROL
repeat
  if query('Please click YES or NO') then writeln('RESPONDED YES') else writeln('RESPONDED NO');
  flash('User interaction'#10'demo');
  writeln(query('JUST LET THIS TIMEOUT (1)',3000,TRUE));
  writeln(query('JUST LET THIS TIMEOUT (2)',3000,FALSE));
until not query('Carry on?');
```

displays a dialog containing YES button and a NO button under the **Caption** **Please click YES or NO**:



If you click YES, **writeln** 37 procedure writes **RESPONDED YES** in the *EscapeE* log but if you click NO, **RESPONDED NO** is written instead.
Another dialog then appears with the caption **JUST LET THIS TIMEOUT** to demonstrate **Timeout** in use.
If you click YES, **TRUE** is logged, if you click NO, **FALSE** is logged. If you do not click the YES or NO buttons then after 3 seconds, **Query** 16 function times-out and **TRUE** is written to the log.
Another timeout dialog appears to demonstrate the **Default** value **FALSE** being logged.
The last dialog appears with **Carry on?** as the **Caption**
Click YES to run through the example again or NO to finish.

Rotate example

Rotate(Angle: number);

In this example, text and digits are shown at a range of angles to form a protractor for text.

```
// REDTITAN RS2 CONTROL

circle(1200,1200,600);
ANGLE:=0;
TANGLE:=inttostr(ANGLE);
N:=15;
ROT:=ANGLE;

repeat
  TANGLE:=inttostr(ROT);
  rotate(ROT);
  text(1200,1200,'.            ..... ' + TANGLE + ' degrees');
  ROT:=ROT+N;
until ROT>350;

halt(1);
```

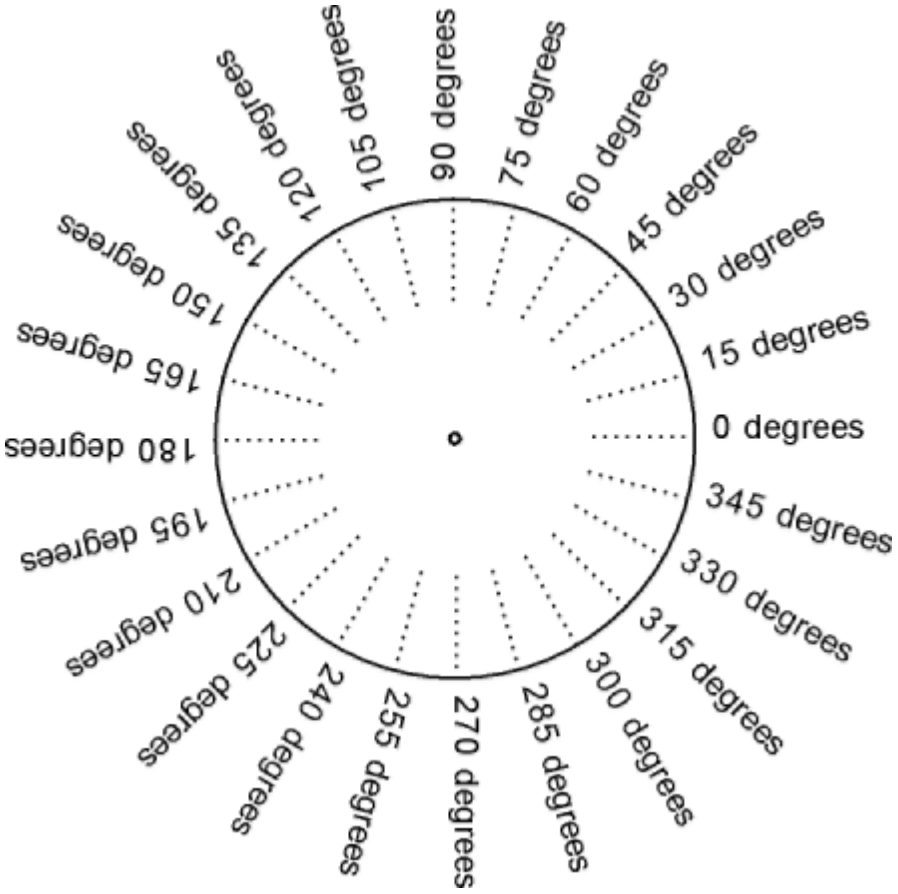


Table example: Basic

Tab(Xpos,Ypos:number;Htabs:list;Vspace:number):tableHandle;

A simple, self-contained, 3-column x 4-row table.

```
// REDTITAN RS2 CONTROL

tx:=tab(300,600,[500,1200,1500],5);
tab_pad(tx,40);

//table headings

font(12,'Arial','bold');
tab_align(tx,'c');

tab_cell(tx,'');
tab_cell(tx,'Telephone');
tab_cell(tx,'email');
tab_nextrow(tx,true);

//table body

font(10,'Arial','');
tab_align(tx,'l');

tab_cell(tx,'UK');
tab_cell(tx,'(+44) [0]1494 811 420');
tab_cell(tx,'help@redtitan.com')
tab_nextrow(tx,true);

tab_cell(tx,'USA');
tab_cell(tx,'770.924.1226');
tab_cell(tx,'help@redtitan.com')
tab_nextrow(tx,true);

tab_cell(tx,'France');
tab_cell(tx,'(+33) [0]2 32 60 20 53');
tab_cell(tx,'RTfrance@RedTitan.fr')
tab_nextrow(tx,true);

tab_cell(tx,'Germany');
tab_cell(tx,'(+49) [0] 30 60985629');
tab_cell(tx,'deutschland@redtitan.com')
tab_nextrow(tx,true);

halt(1);
```

Table example: Variable data

```
Tab(Xpos,Ypos:number;Htabs:list;Vspace:number):tableHandle;
OpenFile(Name:string):fileHandle
ReadCSV(var Handle:fileHandle;var Lx:list;[Separator:string[1]]);
```

You need a separate CSV file to supply the data for this RS/2 file.

```
// REDTITAN RS2 CONTROL

paper('a1',false);

if paramstr(1)='1' then
  begin  // first page only
    filename:=paramstr(2);
    if fileexists(FILENAME) then fx:=openFile(FILENAME) else
      begin
        filename:=browse('CSV file','','CSV files (*.csv)|*.csv|All
files (*.*)|*.*');
        if trim(filename)='' then halt(1);
        fx:=openfile(filename);
      end;
    csv_buffer:=[];
    readcsv(fx,CSV_BUFFER);
    headings:=csv_buffer;
  end;

orient('L');
pen(1);
tx:= tab(200,800,[600],5);
tab_align(tx,'1');
tab_pad(tx,40);

// display headings on every page

text(200,200,Filename);

for i:=0 to pred(length(headings)) do
tab_cell(tx,list_strings(headings,i));

tab_nextrow(tx,true);
repeat
  readcsv(fx,CSV_BUFFER);
  for i:=0 to pred(length(csv_buffer)) do
tab_cell(tx,trim(list_strings(csv_buffer,i)));
  if tab_nextrow(tx)>12600 then break;
until eof(fx);

if eof(fx) then halt(1);
```

WriteLn example

WriteLn(P1 [, P2, ...,Pn]);

This RS/2 script shows how information may be placed in the *EscapeE* log.

```
// REDTITAN RS2 CONTROL

// WriteLn procedure can accept most parameter types, but not unsupported
or structured types (e.g. list).

x:=1; y:=2; b:=(x=y); s:='My string';

writeln('x = ',x,' y = ',y,' b = ',b,' s = ',s);
writeln('Each WriteLn statement is placed ');
writeln('on a separate line.');
```

The resulting Log page shows this text:

```
x = 1 y = 2 b = FALSE s = My string
Each WriteLn statement is placed
on a separate line.
```

Index

/

// comment 5
// REDTITAN RS2 CONTROL 8

A

About RS/2 5
align table text, horizontal 24
align table text, vertical 25
AND, operator 6
angle, Rotate 29
angles, Line example 42
angles, Rotate example 44
ascent, text metrics 29
ASCII 8
assigning values 5
attributes list, Format 27

B

back, Pred function 15
basic table 45
BEGIN, statement 6
Bezier, Curve 23
bins 32
BOOLEAN type 5
borders, table cell 26
Box procedure 22
break, Col_Break 19
BREAK, statement 6
Browse function 16
Brush procedure 21
Button procedure 22

C

CASE, statement 6
cell, table text 24
Char function 12
characters, number of 17
Circle procedure 22
clip region, Resource 34
clip region, ResourcePDF 35
CloseFile procedure 34
Col_Blank procedure 19
Col_Break procedure 19
Col_Compose function 19
Col_Count function 20
Col_Create function 18
Col_Display function 19
Col_Font function 18
Col_Free function 19
Col_Height function 20
Col_Leading procedure 20
Col_Style function 18
Col_Text function 18
Color procedure 21
color, Brush 21
Colour procedure 21
Column example 39
command line 36
comment 5
composite field value 15
console, WriteLn 37
convert
    IntToStr 14
    StrToFloatDef 11
    StrToIntDef 11
Copies procedure 31
Copy function 11
count, lines in column 20
count, parameters 36
create column 18
create RS/2 file 8
CreateFile function 34
CSV, read next record 33
Curve procedure 23
Custom procedure 32

D

date, Now 30
day, Now 30
Dec procedure 13
Delete procedure 11
descent, text metrics 29
dialog

dialog
Browse 16
InputBox 16
ShowMessage 16
DO, statement 6
down, Pred 15
draw
Brush color 21
Button 22
Circle 22
Col_Display 19
Color 21
Curve 23
Frame 23
Line 21
Pen 21
PolyLine 23
Tab_NextRow 25
Text 29
Duplex option 31

E

edit box, dialog 16
elements
about 7
Length 17
List_Add 17
List_Clear 17
List_Numbers 17
List_Strings 17
ELSE, statement 6
Encoding 8
End of file 8
end of file, Halt 36
END, statement 6
EOF function 33
EVALUATE version 5
Even function 13
Examples 39
EXIT, statement 6
Extent procedure 29

F

FieldValue function 15
file
Browse 16
CloseFile 34
CreateFile 34
NewFile 33
OpenFile 33
Resource 34
ResourcePDF 35
FileExists function 34
fill
Box 22
Brush color 21
Button 22
column format 19
Flash procedure 16
floating point, convert to 11
floating point, FormatFloat 28
Font procedure 29
font, column 18
FOR, statement 6
Format function 27
format, column 19
format, table text 26
FormatFloat function 28
Frame procedure 23
FUNCTION, statement 6
functions
list 10
private 5

G

Getting started 8

H

Halt procedure 36
HANDLE 5
height, column 20
home, table cell 25
hour, Now 30

I

Identifiers 6
IF, statement 6
Inc procedure 13
InputBox example 41
InputBox function 16
IntToStr function 14

inverse, Orient 31

J

journal, Orient 31
justify, column 19

L

landscape, Orient 31
Length function 17
length of string 29
line
- blank 19
- break, column 19
- Color 21
- count, column 20
- example 42
- Pen 21
- Polyline 23
- procedure 21
- ReadLine 33
- spacing 20
- width 21
- WriteLine 34
- WriteLn 37

Line example 42
Line procedure 21
LIST type 5
List_Add function 17
List_Clear procedure 17
List_IndexOf function 17
List_Numbers function 17
List_Strings function 17
log, WriteLn 37
Long edge binding 31
LowerCase function 12

M

Max function 14
metrics, Extent 29
millisecond, Now 30
milliseconds, Sleep 36
Min function 14
minute, Now 30
MOD, operator 6
month, Now 30

N

NewFile procedure 33
next row, table 25
next, Succ function 15
NOT, operator 6
Notes on types 7
Now function 30
number
- Col_Count 20
- Copies 31
- IntToStr 14
- page 36
- StrToFloatDef 11
- Type 5

NUMBER type 5

O

Odd function 13
OF, statement 6
OpenFile function 33
Operators 6
OR, operator 6
Ord function 12
Orient procedure 31

P

padding, table cells 25
page number 36
page, Resource 34
page, ResourcePDF 35
paper dart, Line example 42
Paper procedure 31
paper trays and bins 32
ParamCount function 36
ParamStr function 36
path 36
Pen procedure 21
Pipe function 37
Plex procedure 31
PolyLine procedure 23
portrait, Orient 31
Pos function 11
preamble 5

Pred function 15
procedures
list 10
protractor, Rotate example 44

Q

Query example 43
Query function 16

R

radius, Circle 22
Random function 14
ReadCSV procedure 33
ReadLine function 33
rectangle, Frame 23
REPEAT, statement 6
Resource function 34
ResourcePDF function 35
Rotate example 44
Rotate procedure 29
row, Tab_NextRow 25
RS/2 procedures list 10
RS/2 syntax 5
RS2 version 5
RTRS2IN version 5
Runtime environment 5

S

scale, Resource 34
scale, ResourcePDF 35
search, list 17
search, string 11
second, Now 30
section, Circle 22
server, Pipe 37
SetTrays procedure 32
Short edge binding 31
ShowMessage procedure 16
Simplex option 31
size, Custom 32
size, Paper 31
Sleep procedure 36
spacing, Col_Leading 20
Split procedure 12
Statements 6
string
Length 17
List_Add 17
List_Strings 17
Type 5
WriteLine 34
STRING type 5
StrToFloatDef function 11
StrToIntDef function 11
style, column 18
style, Font 29
substring
Copy 11
Delete 11
Pos 11
Succ function 15

T

Tab function 24
Tab_Align procedure 24
Tab_Borders procedure 26
Tab_Cell procedure 24
Tab_Format function 26
Tab_Home function 25
Tab_NextRow function 25
Tab_Pad procedure 25
Tab_VAlign procedure 25
table
align horizontal 24
align vertical 25
basic, example 45
cell borders 26
define 24
first cell 25
formatted text 26
new row 25
padding 25
text 24
variable data, example 46
Table example: Basic 45
Table example: Variable data 46
terminate program 36
text

text
Col_Text 18
Color 21
Extent 29
Font 29
procedure 29
Tab_Cell 24
window 16
WriteLn 37
Text procedure 29
THEN, statement 6
time, Now 30
time, Sleep 36
timeout, Query 16
timeout, ShowMessage 16
TO, statement 6
transparent, Resource 34
transparent, ResourcePDF 35
tray and bin numbers 32
Trim function 11
types 7

U

UNICODE 8
UNTIL, statement 6
up,Succ function 15
UpperCase function 12
UTF8 8

V

value, field 15
variable data, table 46
version 5

W

WHILE, statement 6
whitespace, remove 11
width, Pen 21
write text 29
write text in column 19
WriteLine procedure 34
WriteLn example 47
WriteLn procedure 37

Y

year, Now 30
yes/no dialog 16