WebPascal脚本模型教程 - 基本语法

2017-01-04 10:38:12 99+

在rtc组件的目录下有个脚本演示，建议去看一下，在\QuickStart\rtcScript目录下。

```
脚本需要包含在 <? 和 ?> 之中。

//脚本中的字符串
Strings inside the Script:

    Long HTML strings are shown in HTML editor
    and may be spread across multiple lines:
//这是一个单行字符串的表现形式
    ?> this is a single-line string <?
//这是一个多行字符串的表现形式
        ?> this would be a multi-line string,
           which can spread across
           the whole document <?

    Short strings will NOT show in HTML editors
    and can NOT be spread across multiple lines
    (need to be opened and closed on the same line):
    " abc "
    ' abc '
//字符串用单引号或双引号括起来

//整型为0到9
Characters for Integer values:
    0-9

//浮点型为0到9和小数点
Characters for Floating-point values:
    0-9
    .

//数组，记录和函数调用参数的打开和关闭：
Arrays, Records and Function call parameters opening and closing:
    ( )
    [ ]
    { }
    begin end

//注释
Comments
//单行
    // abc        single-line comment
//多行
    (* abc *)     multi-line comment

//命令和元素分隔符（数组，记录集，函数调用参数）
Command and List element separators (arays, records, function call parameters):
    ,
    ;

//函数和变量名称中允许的字符
Characters allowed in Function and Variable names:
    a..z
    A..Z
    0..9
    _
    $

//用Delphi编写的函数需要由字母或者下划线开始
Functions written in Delphi begin with:
    a..z
    A..Z
    _

//脚本变量和函数以$号开始
Script Variables and Functions begin with:
    $

//使用变量名称作为字符串访问脚本变量
Accessing Script variables using a variable name as string:
//执行表达式以获取变量名称，返回变量内容
```

```
    $( expression ) - executes the expression to get variable name, returns variable content

    Examples://示例
    $('x') is the same as $x //$('x')和$x是一样的
    $('x.y') is the same as $x.y
    $y:="x"; $($y) - since variable $y is "x", $($y) returns the value of $x

//可以读取和更改变量（分配新值）。
    Variables can be read and changed (new values assigned) like this.

//访问数组中的字段
Accessing fields in an Array:
    .123        - get element at index 123
    ( 123 )     - get element at index 123

    ( expression )  - get element at index we get after evaluating the expression
              Expression will be converted to an Integer

//访问记录集中的字段
Accessing fields in a Record:
    .XYZ        - get field with name "XYZ"
    ('XYZ')     - get field with name "XYZ"

    ( expression )  - get element with the name we get after evaluating the expression
              Expression will be converted to a string if it isn't a string

//访问数据库记录集中的字段
Accessing fields in a DataSet:
    .123        - get field with index 123
    .XYZ        - get field with name "XYZ"
    ( 123 )     - get field with index 123
    ('XYZ')     - get field with name "XYZ"

    ( expression )  - if expression starts with a digit, get field at given index.
              Otherwise, get field with given name.

//数组变量上可用的属性
Properties available on Array variables:
    .COUNT      - number of elements in array
    .FIELDCOUNT - same as .COUNT

//记录集变量上可用的属性
Properties available on Record variables:
    .COUNT      - number of fields in record
    .FIELDCOUNT - same as .COUNT

//数据库记录集变量上可用的属性
Properties available on DataSet variables:
    .COUNT      - number of rows in dataset
    .ROWCOUNT   - same as .COUNT
    .FIELDCOUNT - number of fields in row
    .ROW        - current row number (read/write property: set to change row)

    .EOF        - returns TRUE if behind last row
    .BOF        - returns TRUE if before first row
    .EMPTY      - returns TRUE if dataset is empty (ROWCOUNT = 0)

//DataSet变量上可用的方法（方法返回NULL）
Methods available on DataSet variables (methods return NULL):
    .FIRST      - jumps to first row
    .LAST       - jump to last row
    .NEXT       - jumps to the next row
    .PRIOR      - jumps to the prior row

    .INSERT     - inserts a new row at the current positon
    .APPEND     - appends a new row at the end of the dataset
    .DELETE     - deletes the current row from the dataset

//函数调用
Function calls:
//无参数
    Without parameters ...

    FunctionName
    FunctionName()

    $FunctionName
    $FunctionName()

//有参数
```

```
    With parameters sent inside the "PARAMS" array:

    FunctioName (value1, value2, ...)
    FunctioName (value1; value2; ...)

    $FunctioName (value1, value2, ...)
    $FunctioName (value1; value2; ...)

//也可以带参数名
    With parameters sent using parameter names:

    FunctionName (paramName1:value1, paramName2:value2, ...)
    FunctionName (paramName1:value1; paramName2:value2; ...)

    $FunctionName (paramName1:value1, paramName2:value2, ...)
    $FunctionName (paramName1:value1; paramName2:value2; ...)

//构建数组
Constructing Arrays
    (value1, value2, value2 ...)
    (value1; value2; value2 ...)

//构建记录集
Constructing Records
    (name1:value1, name2:value2, name3:value3)
    (name1:value1; name2:value2; name3:value3)

//为空的表达形式
NULL or NILL values (the same effect):
    NULL
    NIL

//构建变量方式
Constructor:
    NEW ARRAY
    NEW RECORD
    NEW DATASET
    NEW STREAM  or NEW BYTESTREAM
    NEW INTEGER or NEW INT  or NEW INT32
    NEW LARGEINT    or NEW INT64
    NEW DATETIME    or NEW DATE or NEW TIME
    NEW FLOAT   or NEW DOUBLE
    NEW CURRENCY
    NEW BOOLEAN or NEW BOOL
    NEW WIDESTRING
    NEW TEXT

    Example:
    $x := new DataSet;

//操作符
Left-side operators, working as functions with 1 parameter:

    not not X

    !   ! X - same as not X
    -   - X

    #   # X

String, numeric and logical operators for functions with 2 parameters:

    +   X + Y
    -   X - Y
    #   X # Y   - X is string, Y is

    div X div Y
    mod X mod Y

    *   X * Y
    /   X / Y   - same as X div Y
    %   X % Y   - same as X mod Y

    shl X shl Y
    shr X shr Y

    <<    X << Y    - same as X shl Y
    >>    X >> Y    - same as X shr Y

    and X and Y
```

```
    or  X or Y
    xor X xor Y

    &   X & Y   - same as X and Y
    |   X | Y   - same as X or Y

    { not implemented }
    ^   X ^ Y


Comparison operators for functions with 2 parameters:
    >    X > Y
    <    X < Y
    =   X = Y
    ==  X == Y  - same as X = Y

    <=   X <= Y
    >=   X >= Y

    <>    X <> Y
    !=  X != Y  - same as X <>  Y

    { not implemented }
    in  X in Y
    is  X is Y

Assignment operators:
    :=  X := Y
    +=  X += Y
    -=  X -= Y
    *=  X *= Y
    /=  X /= Y
    %=  X %= Y
    &=  X &= Y
    |=  X |= Y

    { not implemented }
    ^=  X ^= Y

//操作符优先级
Operator priority:

    Assignment operators always evaluete the complete right side
    before assigning the result to the variable on the left side.

    High priority operators:
    * / % div mod

    High priority operators work on the result from the left side
    and a the first element from are right side.

    Low priority operators:
    < > <= >= == != <>

    Low priority operators work on the result from the left side
    and the result from the right side (executed as last command).

//在脚本中编写函数
Writing Functions inside Script:

    FUNCTION $FunctionName command;

    Function result has to be assigned to the $RESULT variable,
    which can also be used to store temporary data until the final result
    has been prepared. Once the function execution completes, only the data
    stored in the $RESULT variable will be passed on as function result.

    FUNCTION $FunctionName $result := command;

    If the function has more than a single command, put it isnide begin/end.
    Here is an example of a function to return a sum from 1 to the passed parameter:

    FUNCTION $test if $params <> nil then
      begin
      $result := 0;
      for $i := 1 to $params.0 do $result += $i;
      end;

    And here is how this simple function can be called:
```

```
    $test(100);

//条件表达形式
Conditional code branching:

    IF condition THEN command;

    IF condition THEN command1 ELSE command2;

//循环
Command Loops:

    REPEAT command-list UNTIL condition;

    WHILE condition DO command;

    FOR $variable := minvalue TO maxvalue DO command;
    FOR $variable := maxvalue DOWNTO minvalue DO command;

   { not implemented }
    FOREACH $variable := enumeration DO command;

//代码块
Code block declaration:

    CODE command

    Code Blocks are "blocks" of execution code which you can assign to local
    script variables or send as parameters to local script function calls.

    When a code block (command with a "code" prefix) is passed to a script function call,
    it will be executed in the scope of the script function.

    When a code block is assigned to a variable (or a variable property),
    each time the variable is evaluated, the code block (assigned to that variable)
    will be executed and the result returned as if the code was called as a function and
 stored into the variable.

    Examples:

    // define a code block for variable $x ...
    $x := code if $y>0 then $y else -$y;

    $y := 15;
    $x; // execute code x" output 15

    $a := code $x; // $a will be returning the value of $x

    // you can pass codeblocks to local function calls ...
    $myFunc(a:code $x);

    // you can pass any expression as a code block ...
    $myFunc(code if $x>0 then $x else $y);



Code block access:

    // To assign a codeblock to a variable, use ...

    $x := code $y;

    @x := 15;//this will assign "15" to variable $y, since it is stored in codeblock x

    $x := 25; // this will assign value "25" to variable $x,
 replacing any prior assignments to $x



NOTE: Function calls can be nested, all operators can be used anywhere inside a script.

Results from any function call, command and variables which are not assigned
to other variables will implicitly become part of the resulting output (HTML/XML).

All variable and function names are case-insensitive (NOT CASE SENSITIVE),
so you can use lowercase, uppercase or a combination of lower and uppercase names.

注意：函数调用可以嵌套，所有操作符都可以在脚本中的任何地方使用。

来自任何未分配的函数调用，命令和变量的结果
```

```
到其他变量将隐式地成为结果输出（HTML / XML）的一部分。

所有变量和函数名称不区分大小写（NOT CASE SENSITIVE），
因此您可以使用小写，大写或小写和大写名称的组合。


//其它一些对象的属性
Request parameters can be read and written using the "Request" variable:

    REQUEST.METHOD      = request method (examples: "GET", "POST", "PUT", ...)
    REQUEST.FILENAME    = request file name (example: "/myfile.rtc" )
    REQUEST.CLOSE       =should the connection be closed after respose was sent?(HTTP/1.0)
    REQUEST.URI     = Request URI (all exept the host)
    REQUEST.URL     = Request URL (host + URI) - read only

    REQUEST.CONTENTTYPE = "CONTENT-TYPE" HTTP reuqest header value
    REQUEST.CONTENTLENGTH   = "CONTENT-LENGTH" HTTP request header value
    REQUEST.HOST        ="HOST" HTTP request header value(example:www.realthinclient.com)
    REQUEST.AGENT       ="USER-AGENT" HTTP request header value(example:Mozila Firefox...)
    REQUEST.REFERER     = "REFERER" HTTP request header value
    REQUEST.FORWARDEDFOR    = "X-FORWARDED-FOR" HTTP request header value


    REQUEST.HEADER      or
    REQUEST.HEADER.TEXT = complete HTTP request header

    REQUEST.HEADER('abc')   or
    REQUEST.HEADER.abc  = value of the HTTP request header with name 'abc'
(example name: "CONTENT-TYPE")

    REQUEST.HEADER.COUNT        = number of HTTP request header variables
    REQUEST.HEADER.NAME( 123 )  =name of HTTP request header at index 123(starting at 0)
    REQUEST.HEADER.VALUE( 123 ) =value of HTTP request header at index 123(starting at 0)


    REQUEST.COOKIE      or
    REQUEST.COOKIE.TEXT = complete request cookie text

    REQUEST.COOKIE('abc')   or
    REQUEST.COOKIE.abc  = value of request cookie parameter with name 'abc'

    REQUEST.COOKIE.DELIMITER    = request cookie parameters delimiter
    REQUEST.COOKIE.COUNT        = number of parameters in a request cookie - read only
    REQUEST.COOKIE.NAME( 123 )
= name of the request cookie parameter at index 123 (starting at 0)
    REQUEST.COOKIE.VALUE( 123 )
= value of the request cookie parameter at index 123 (starting at 0)


    REQUEST.QUERY       or
    REQUEST.QUERY.TEXT  = complete request query text

    REQUEST.QUERY('abc')    or
    REQUEST.QUERY.abc   = value of request query parameter with name 'abc'

    REQUEST.QUERY.DELIMITER     =request query parameter delimiter (examples: ";", "&")
    REQUEST.QUERY.COUNT     = number of query parameters - read only
    REQUEST.QUERY.NAME( 123 )   = name of query parameter at index 123 (starting at 0)
    REQUEST.QUERY.VALUE( 123 )  =value of query parameter at index 123 (starting at 0)


    REQUEST.PARAMS      or
    REQUEST.PARAMS.TEXT = complete input as one string

    REQUEST.PARAMS('abc')   or
    REQUEST.PARAMS.abc  = value of input parameter with name 'abc'

    REQUEST.PARAMS.DELIMITER    = input parameters delimiter
    REQUEST.PARAMS.COUNT        = number of input parameters - read only
    REQUEST.PARAMS.NAME( 123 )= name of input parameter at index 123 (starting at 0)
    REQUEST.PARAMS.VALUE( 123 )= value of input parameter at index 123 (starting at 0)


    REQUEST.INFO('abc') or
    REQUEST.INFO.abc    = access to "Request.Info" variable 'abc'
                  All Request.Info variables can be accessed from Delphi.

    REQUEST('abc')      or
    REQUEST.abc     = Value of the HTTP request header with name "abc"
                  Example: Request("CONTENT-TYPE")
```

```
471
472
473
474 Response parameters can be read and written using the "Response" variable:
475
476
477     RESPONSE.STATUS
478 = response Status Code+Text("200 OK", "404 File Not Found", etc)
479
480     RESPONSE.STATUSCODE or
481     RESPONSE.STATUS.CODE    = response Status Code (200, 404, ...)
482
483     RESPONSE.STATUSTEXT or
484     RESPONSE.STATUS.TEXT    = response Status Text ("OK", "File Not Found", etc)
485
486     RESPONSE.CONTENTTYPE    = "CONTENT-TYPE" HTTP response header value
487     RESPONSE.CONTENTLENGTH  = "CONTENT-LENGTH" HTTP response header value
488
489
490     RESPONSE.HEADER     or
491     RESPONSE.HEADER.TEXT    = complete HTTP response header
492
493     RESPONSE.HEADER('abc')  or
494     RESPONSE.HEADER.abc = value of the HTTP response header with name 'abc'
495                   (example name: "CONTENT-TYPE")
496
497     RESPONSE.HEADER.COUNT       = number of HTTP response header variables
498     RESPONSE.HEADER.NAME( 123 ) = name of HTTP response header at index 123
499  (starting at 0)
500     RESPONSE.HEADER.VALUE( 123 )    = value of HTTP response header at index 123
501  (starting at 0)
502
503
504     RESPONSE.COOKIE     or
505     RESPONSE.COOKIE.TEXT    = complete response cookie text
506
507     RESPONSE.COOKIE('abc')  or
508     RESPONSE.COOKIE.abc = value of response cookie parameter with name 'abc'
509
510     RESPONSE.COOKIE.COUNT       =number of parameters in a response cookie-read only
511     RESPONSE.COOKIE.NAME( 123 )
512 = name of response cookie parameter at index 123 (starting at 0)
513     RESPONSE.COOKIE.VALUE( 123 )
514 = value of response cookie parameter at index 123 (starting at 0)
515     RESPONSE.COOKIE.DELIMITER   = response cookie parameters delimiter
516
517
518     RESPONSE('abc')     or
519     RESPONSE.abc        = Value of the HTTP response header with name "abc"
520                   Example: Response("CONTENT-TYPE")
521
522
523 You can work with sessions by using the "Session" variable:
524
525     SESSION.ID      = Session ID
526
527     SESSION.OPEN        or
528     SESSION.OPEN.FWDLOCK    or
529     SESSION.OPEN.PRIVATE    = Open and lock a new private user session.
530                   This is the default private session type,
531                 which works with users behind all kinds of Proxy servers.
532                   Locks the Session to current user's IP address or to
533                   the "X-FORWARDED-FOR" HTTP header (if it exists).
534
535     SESSION.OPEN.PUBLIC = Open and lock a new PUBLIC Session, which should NOT be locked
536  to this specific user.
537               This Session can be accessed by any user who knows the Session ID.
538                   When working with PUBLIC sessions, always use SESSION.LOCK() to
539  obrain a lock.
540
541     SESSION.OPEN.IPLOCK = This option will NOT work for users behind Proxy Servers with
542  changing IP.
543                   Lock the Session to users's IP address,
544  ignoring "X-FORWARDED-FOR" HTTP header.
545
546     SESSION.OPEN.SECURE or
547     SESSION.OPEN.STRONG or
548     SESSION.OPEN.IPFWDLOCK  = This option will NOT work for users behind Proxy Servers with
549  changing IP.
550                   Opens and locks a new session with a strong and secure link to
```

```
 the current user.
                  Lock the Session to users's IP address *and* to the
"X-FORWARDED-FOR" HTTP header.

   SESSION.FIND('abc') or
   SESSION.FIND.abc    = Default method for finding and locking Private user Sessions.
                  Finds a Session with ID "abc" and tries to lock it,
 without waiting.
                  Returns FALSE if Session can not be locked because does
 NOT exist,
                    or if it is locked by anotuer user.
                  Returns TRUE and Locks the Session if Session exists and
                    is NOT currently locked by another user.

   SESSION.HAVE('abc') or
   SESSION.HAVE.abc    = Check if session with ID "abc" exists.
                  Returns TRUE if Session exists (even if locked), but does
 NOT lock the session.

   SESSION.LOCK('abc') or
   SESSION.LOCK.abc    = Lock a PUBLIC Session (which can be shared by multiple users).
                  Find Session with ID "abc" and wait until you can lock it
 for usage.
                  If session exists, it will be locked and TRUE will be returned.
                  If Session does NOT exist
 (or should be closed while waiting for a lock), returns FALSE.

   SESSION.CLOSE       = close current session

   SESSIOn.CLOSE('abc')    or
   SESSIOn.CLOSE.abc   = Find and close session 'abc'

   SESSION.KEEPALIVE   =number of seconds the session should be kept alive when not used
   SESSION.FINALEXPIRE = date/time when the Session has to expire, regardless
 of "SESSION.KEEPALIVE"

   SESSION.EXPIRETIME  = date/time when the Session would expire if not used (read-only)

   SESSION.COUNT       or
   SESSION.COUNT.TOTAL = Total number of currently active sessions (read-only)

   SESSION.COUNT.LOCKED    = Number of currently active and locked sessions (read-only)
   SESSION.COUNT.UNLOCKED  = Number of currently active but not locked sessions (read-only)

   SESSION('abc')  or
   SESSION.abc= access to current sessions variable "abc". Before you can access session
 variables,
              you need to find and lock an existing session using "Session.Find"
(or from a Delphi function),
              or open and lock a new session using "Session.Open". If no session
 is locked,
              all variables will return NULL on read access, but an exception will
 be raised on write access.


Data received through as Query parameters (part of the URL) can be accessed
by using the REQUEST.QUERY variable, or by using the QUERY variable
 (without the "REQUEST." prefix):

   QUERY           or
   QUERY.TEXT      = complete request query text

   QUERY.COUNT     = number of query parameters - read only
   QUERY.DELIMITER     = request query parameter delimiter (examples: ";", "&")
   QUERY.NAME( 123 )   = name of query parameter at index 123 (starting at 0)
   QUERY.VALUE( 123 )  = value of query parameter at index 123 (starting at 0)

   QUERY('abc')        or
   QUERY.abc       = value of request query parameter with name 'abc'


Data received from a FORM POST when using INPUT elements inside HTML can be accessed
 through the INPUT variable:

   INPUT           or
   INPUT.TEXT      = complete input as one string

   INPUT.COUNT     = number of input parameters - read only
   INPUT.DELIMITER     = input parameters delimiter
   INPUT.NAME( 123 )   = name of input parameter at index 123 (starting at 0)
```

```
    INPUT.VALUE( 123 )  = value of input parameter at index 123 (starting at 0)

    INPUT('abc')        or
    INPUT.abc       = value of input parameter with name 'abc'


Basic Client information can be obrained by using the CLIENT variable (read-only):

    CLIENT      = Client's IP address and Port in the form "IP:Port"
 (example: "196.45.32.112:1604")

    CLIENT.IP   or
    CLIENT.ADDR or
    CLIENT.ADDRESS  = Client's IP address

    CLIENT.PORT = Client's Port number (as string)

    CLIENT.COUNT    =Number of currently conencted clients(total client connection count)


Basic Server information can be obtained by using the SERVER variable (read-only):

    SERVER      = Server's IP address and Port in the form "IP:Port"
(example: "127.0.0.1:80")

    SERVER.IP   or
    SERVER.ADDR or
    SERVER.ADDRESS  = Our Server's Local IP address

    SERVER.PORT = Out Server's Port number (as string)
              Standard HTTP port = 80, Standard HTTPS (SSL) port = 443
```