We will like to thank Myrna Goran and the rest of his family for allowing this book to be released to the public.

Dick K. Goran was a prolific REXX developer and we also want this ebook to be a tribute to his work.

Richard K. Goran (1942-1999)

OS/2® WARP

R E X X Reference Summary Handbook

"Everything You Wanted To Know About Managing Workplace Shell Objects With REXX, But Didn't Know Where To Look"

Plus: OS/2 Version 2.1 REXXUTIL & *REXXLIB* WPTools & RxFTP

REXX

Reference Summary Handbook

Thank you: HES

I want to extend my gratitude to those people who provided assistance and reviewed the Handbook for technical accuracy: Evan Bergman and Gregory Czaja (Development Technologies, Inc.); Tom Bridgman; Charles Daney (Quercus Systems); Rick McGuire and Steve Price (formerly of IBM REXX Development); Larry Martin (Carry Associates); David Moskowitz (Productivity Solutions).

My thanks also to Mike Cowlishaw of IBM Hursley, the "father" of REXX, for his kind words about this Handbook.

C F S Nevada, Inc.

953 E. Sahara Avenue, Suite 9B
Las Vegas, Nevada 89104-3012

Voice	702-732-9616
FAX	702-732-3847
Internet	rrsh@cfsrexx.com
World Wide Web	http://www.cfsrexx.com

Richard K. (Dick) Goran,
President & CEO
dgoran@cfsrexx.com

Fourth Edition
ISBN: 0-9639854-3-4
SRL & PUBORDER S246-0078-2

Changes in this edition are denoted by a vertical bar in the left margin. When the changes include the addition of an entire section, the changes are not marked.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1. Language Summary

The information provided in this handbook has been collected from multiple sources. It is intended to serve as a "keyboard-side" aid in the use of REXX under IBM's OS/2 2.x and OS/2 Warp Versions 3 and 4. Items introduced with Warp Version 3 are indicated by **(V3)**. Items introduced with Warp Version 4 (originally code named *Merlin*) are indicated by **(V4)**. Items made obsolete by Warp Version 3 are indicated by **(pre V3)**. Items available in Object REXX only are indicated by **(OBJ)**.

A complete reference for OS/2 SAA REXX (referred to as Classic REXX) can be found in the OS/2 Information Folder and in the following IBM Publications: *OS/2 Procedures Language 2/REXX Reference* (IBM publication number S10G-6268) and *OS/2 Procedures Language 2/REXX User's Guide* (IBM publication number S10G-6269). Beginning with Warp Version 4, the online REXX reference (REXX.INF) pertains to the active REXX interpreter with the alternate .INF file being renamed. If Classic REXX is active, the \OS2\BOOK directory will contain REXX.INF and OREXX.INF. If Object REXX is active, \OS2\BOOK will contain CREXX.INF and REXX.INF.

Information for the use of the external REXX function packages included in this publication was obtained from the provider of the respective modules. Comments or questions regarding the use of these external REXX function packages should be directed to those sources.

Numerous OS/2 utilities written in REXX by the author are available free of charge via anonymous FTP from *ftp.cfsrexx.com/pub/* or with a Web browser on the World Wide Web at *http://www.cfsrexx.com*.

The following syntax is be used throughout this publication:

[item]
Item is optional and is entered without the brackets.

{<u>one</u> | two}
A selection of either *one* or *two* must be made. Default selections are underlined.

Alphabetic case, except within literal strings, is transparent unless otherwise noted.

Lines which would normally appear as a single line but are split here because of the size constraint of the format of this publication appear as:

```
    Beginning of long line →
                          → remainder of long line
```

° indicates leading zeroes are suppressed when a numeric value is returned.

1.1 Basic Fundamentals and Structure

A REXX program contains clauses built from combinations of the tokens defined below and terminated with a ";" (semicolon); however, the semicolon can be omitted at the end of a line as it is implied by the line end character(s) - normally a carriage return ('0D'x) and line feed ('0A'x).

Comments

`/* this is a comment */`

Each REXX procedure / program, in the form of a .CMD file, must begin with a comment which starts in column 1 of line 1.

Literal strings

Characters enclosed in single (') or double (") quotes. A trailing 'X' specifies hexadecimal notation. A trailing 'B' specifies binary notation. Examples:

```
'Hello!'    "O'Leary's cow"    'Ryan''s Express'
"C1"x       '1010'b
```

Note: *There is an implementation-defined limit of 250 characters for a literal string.*

Symbols

Composed of characters from the following group:

```
                a-z  A-Z  0-9  . ! ? _
```

Any lower case alphabetic character in a symbol is translated to uppercase before use.

If a symbol begins with either a digit or a period, it can end with the sequence "E" (or "e") followed immediately by one or more digits.

The purpose of a symbol varies depending on the context in which it is used.

Note: *Care must be used when converting REXX programs from other platforms to OS/2 since many of these other REXX interpreters permit other, non-SAA characters to be used. Of particular note are the characters '@', '$', and '#' which are not permitted in SAA REXX; however, Personal REXX for OS/2 will accept them.*

Variables

Symbols whose value can be changed during the course of execution of a REXX program. Variables that have not been assigned a value (i.e. they are uninitialized) have a default value of the variable name in uppercase. Compound variables use a period ('.') to separate their component parts which include the stem (all characters up to, and including, the first period) and the tail (all characters following the first period).

Labels

Any single symbol followed by a colon. Labels identify the object of Call and Signal instructions and internal function calls.

Numbers

Integer, floating point or exponential notation. Examples:

```
42   -22.78   3.14159625   1.68e+6   1.686E-6
```

Operators and their precedence:

`+,   -,   ¬ ('AA'x)  or   \`

prefix operators: plus, minus, not

`**`

exponentiate (raise to a power)

`*   /   //   %`

multiply (*), divide (/), divide and return only remainder (//), integer divide (%)

`+    -`
add, subtract

`(blank)   ||   (abuttal)`
concatenate: with a blank, without a blank, withou blank

`=   \=   >   <   \>   \<   >=   <=   <>   ><`
normal comparisons

`==   \==   >>   <<   \>>   \<<   >>=   <<=`
strict comparisons

`&`
AND

`|    &&`
OR, exclusive OR

`~` (OBJ
The tilde character is used in Object REXX as th message send character.

Expressions
Any number of literal strings, variables, numbers o function calls, separated by operators and parenthesis.

Function calls

```
return = function_name([expression] →
                              → [, [expression]])
                         or
Call function_name [expression][, [expression]]
```

Invokes *function_name* with *expression* passed a argument string(s). Up to 20 *expressions* are allowed When a function is explicitly called rather than usec as an expression, the special variable RESULT i assigned the function's return value as opposed tc *return* being assigned the value returned by the function. When a function call is used as ar expression (as in the first construct above), the lef parenthesis must abut *function_name*; however, a comment can be placed between *function_name* anc the left parenthesis.

Templates

A list of symbols separated by blanks or patterns which include:

variable name
- the name of a variable to be assigned a value

literal
- used to match within the input string

(variable name)
- variable whose value is used to match the input string

(period)
- a placeholder that receives part of the input string, except that no assignment is actually performed

integer
- absolute character position in the input string

=integer
- same as preceding

+integer
- relative position in the input string

-integer
- same as preceding

=(variable name)
- variable whose value specifies an absolute character position

+(variable name)
- variable whose value specifies a relative character position

-(variable name)
- same as preceding

In addition, a comma can be used in the template for PARSE ARG to indicate that the next argument becomes the input string for the following portion of the template.

1.2 Keyword Instructions

expression

Expression is evaluated and then passed as a command (i.e. passed to the external environment). The special variable **RC** is set to the return code from the command

variable = expression;

Expression is evaluated and the result is assigned to *variable*. There are 3 special variables (i.e. variables that can be set automatically by the program) - RC, RESULT and SIGL. Their use is shown where appropriate.

ADDRESS [environment [expression]]
ADDRESS [VALUE] expression

Redirects the destination of all commands (or single command *expression*) to *environment*, or with *VALUE* to have *expression* evaluated as the *environment*.

The default value of *environment* for OS/2 is CMD (referring to CMD.EXE). The current value of *environment* can be retrieved with the ADDRESS() function (page 16).

ARG [template]

Translates arguments from a function or subroutine call to uppercase and parses them according to the *template* (page 5). Short form of PARSE UPPER ARG.

CALL name [expression][, [expression]]...

Calls the internal routine, built-in function, or external routine *name*, passing each *expression* as an argument. The special variable RESULT will be set by the called routine if an expression is present on the RETURN instruction. If no expression is returned by the called routine, RESULT will become uninitialized

`CALL {ON | OFF} condition [NAME label]`

Activates or deactivates the user-defined *condition* related handlers. Control transfers to the *label* matching *condition* by a CALL instruction if the condition occurs while the trap is ON. *Label* can specify an alternative label name.

Condition can be 'ERROR', 'FAILURE', 'HALT', 'NOTREADY'.

```
DO  [repetitor] [conditional]
    [instruction_list]
END [control_variable]
```

Groups instructions together and optionally repeats them, where *repetitor* is one of:

```
name = expr [TO expr] [BY expr] [FOR expr]
FOREVER
expr
```

conditional is either of:

```
WHILE expr
UNTIL expr
```

instruction_list is any sequence of instructions; *expr* evaluates to a number; and *control_variable* is the name of the respective control variable used in the *repetitor*.

`DROP name [name]...`

Drops (resets to uninitialized state) the variable(s). If *name* is enclosed in parentheses, it is treated as a list of names and variables to drop. If *name* is a stem, then all variables of that stem are dropped.

```
Example:     a = 'x y z'
             drop (a)  /* drops a, x, y, & z */
```

`EXIT [expression]`

Leaves the program, returning *expression* to the caller. *Expression* can be any value; however, some programs which call REXX programs cannot process a return value unless *expression* evaluates to a signed integer in the range (-2^{15} to 2^{15} -1). CMD.EXE is an example of a calling program subject to this restriction.

IF expression[;] THEN[;] instruction
[ELSE[;] instruction]

Expression must evaluate to either 0 or 1. If *expressio* evaluates to '1', the *instruction* following the THEN i executed; otherwise, the *instruction* following the ELS is executed. *Instruction* may be a group of instruction bounded by a DO / END pair.

INTERPRET expression

Evaluates *expression* and then executes it as a instruction.

ITERATE [repetitor]

Starts the next iteration of the innermost or specifie repetitive DO loop as if the END instruction had bee reached. A particular DO loop is identified by i *repetitor*.

LEAVE [repetitor]

Leaves the innermost or specified repetitive DO loop. particular DO loop is identified by its *repetitor*.

NOP

A dummy instruction; does nothing. It is frequently use as the target of a **THEN** or **ELSE** clause.

NUMERIC DIGITS [9 | expression]

Specifies arithmetic precision to *expression* significa digits - default is 9.

NUMERIC FORM [SCIENTIFIC | ENGINEERING]
NUMERIC FORM [VALUE] expression

Specifies the form of exponential numbers. The FOR is set directly by the sub-keywords SCIENTIFIC ENGINEERING or by evaluating *expression*.

NUMERIC FUZZ [expression]

Specifies that *expression* digits, at full precision, are to ignored during numeric comparisons.

OPTIONS expression

Expression is evaluated for [{ETMODE | NOETMODE}] [{EXMODE | NOEXMODE}] to control double-byte character set (DBCS) interpretation. These settings are not applicable to Personal REXX.

PARSE [UPPER] ARG [template]

Parses the arguments according to *template* (page 5) from a function or subroutine call, optionally first translating them to uppercase.

PARSE [UPPER] LINEIN [template]

Parses the input from the default character input stream according to *template* (page 5), optionally first translating it to uppercase.

PARSE [UPPER] PULL [template]

Parses the next line in the REXX data queue according to *template* (page 5), optionally first translating it to uppercase. If the queue is empty, lines will be read from the standard input stream (normally the keyboard).

PARSE [UPPER] SOURCE [template]

Parses the program's source information (3 tokens) according to *template* (page 5), optionally first translating it to uppercase.

```
Example:     OS/2  COMMAND        C:\OS2\REXXTRY.CMD
             OS/2  SUBROUTINE     D:\OS2\rexxtry.CMD
```

Note: *If issued within a subroutine, the information reflects the parent.*

PARSE [UPPER] VALUE [expression] WITH [template]

Parses the value of *expression* according to *template* (page 5), optionally first translating it to uppercase.

PARSE [UPPER] VAR name [template]

Parses the value of *name* according to *template* (page 5), optionally first translating it to uppercase.

PARSE [UPPER] VERSION [template]

Parses the information describing the language processor and level followed by its date, according to *template* (page 5), optionally first translating it to uppercase.

```
Example:
     REXXSAA             4.00  08  Jul  1992
     REXXSAA             4.00  10  Feb  1994      (V3)
     REXXSAA             4.00  24  Aug  1996      (V4)
     OBJREXX             6.00  12  Jul  1996     (OBJ)
     REXX/Personal       4.00  12  Oct  1994
```

PROCEDURE [EXPOSE name [name]...]

Provides a mechanism to protect local variables within an internal routine, and optionally specifies global variables to be exposed (unprotected). If *name* is enclosed in parentheses, it is treated as a list of names of variables to expose.

```
Examples: a = 'x y z'

     label: PROCEDURE EXPOSE (a)  /* will
                 result in x, y, & z being exposed */

     label: PROCEDURE EXPOSE b.   /* will
                 result in all variables with a stem
                 of  b.  being exposed */
```

PULL [template]

Translates the next line in the currently active REXX data queue to uppercase and parses it according to *template* (page 5). If the queue is empty, a line will be read from the standard input stream (normally the keyboard). Short form of PARSE UPPER PULL.

PUSH [expression]

Places the value of *expression* at the top (/LIFO) of the currently active REXX data queue. If *expression* is omitted, a null string is stacked.

QUEUE [expression]

Places the value of *expression* at the bottom (/FIFO) of the currently active REXX data queue. If *expression* is omitted, a null string is stacked.

RETURN [expression]

Returns control, and optionally a value - *expression*, to the caller of the routine or program.

Return to a subroutine (from CALL) causes *expression* to be evaluated and placed in the special variable RESULT. If *expression* is omitted, RESULT is uninitialized (i.e. DROPped).

Return to a function invocation requires *expression* to be specified and that value is then used in the original expression where the function evaluation was invoked.

If a RETURN instruction is executed within a routine that was not invoked by either a CALL instruction or function invocation, RETURN functions like the EXIT instruction.

SAY [expression]

Displays the value of *expression*, appended with a carriage return / line feed character pair ('0D0A'x), on STDOUT (which can be redirected).

```
SELECT
   WHEN expression[;] THEN[;]
     instruction
  [WHEN expression[;] THEN[;]
     instruction]...
  [OTHERWISE[;]
     [instruction]...]
END
```

Selects and executes the first *expression* that evaluates to a '1' and executes its corresponding *instruction*. *Instruction* may be a group of instructions bounded by a DO / END pair.

If none of the *expressions* evaluates to a '1', then the *instructions* following the OTHERWISE (which should always be there) are executed.

SIGNAL label_name
SIGNAL [VALUE] expression

Transfers control to the instruction labeled *label_name*; or evaluates *expression* and transfers control to the instruction labeled with that value.

When control reaches the specified label, the line number of the SIGNAL instruction is assigned to the

special variable SIGL. This can aid debugging becau
the SIGL value can be used to determine the source
a jump to a label.

SIGNAL {ON | OFF} condition [NAME label]

Activates or deactivates the *condition* handler. Conti
transfers to the *label* matching *condition* by a SIGNA
instruction if the condition occurs while trap is ON.

Label can specify an alternative label name. *Conditi*
can be:

ERROR	An external command returned a nc zero return code that either did not resi in a FAILURE condition or that result in a FAILURE condition but a SIGNA ON FAILURE was not active.
FAILURE	An external command failed and result in a return code being passed back to t parent program.
HALT	Results from a <Ctrl-Break> interrupti or some other manual action that caus the program flow to be interrupted.
NOTREADY	Results from a reference to external I/O device that is not reac
NOVALUE	Results from referencing a variable th has not been initialized (not assigned value).
SYNTAX	Results from an incorrect construct reference to an external function wi invalid parameters.

When control reaches the specified label, the li
number of the SIGNAL instruction is assigned to t
special variable SIGL. This can aid debugging becau
the SIGL value can be used to determine the source
a jump to a label.

TRACE [[?] {A | C | E | F | I | L | N̲ | O | R}]
TRACE [VALUE] expression

Controls tracing of program execution (null restores the default). Trace output is written to STDERR.

Trace output can be redirected to *file_name* by appending 2> file_name to the command which invokes the REXX program.

VALUE evaluates *expression* as the trace setting.

?	turns interactive debugging on and off (pause after each instruction), with trace output controlled by the next character. If STDERR is redirected, tracing continues uninterrupted by the pause, effectively ignoring the *?* option.
A	(All) traces all clauses.
C	(Commands) traces all commands.
E	(Error) traces commands with non-zero return codes.
F	(Failure) traces a host command that fails, also indicates the return code.
I	(Intermediates) traces intermediate expression evaluation, results and name substitution.
L	(Labels) traces labels.
N	(Normal) only host commands, after a failed execution. This is the default setting.
O	(Off) Resets tracing to off.
R	(Results) traces all clauses and expressions are traced before execution.

All trace output is prefixed by three characters which identify the type of trace output line:

- Identifies the source of a single clause (i.e. the actual program data).

+++ Identifies a trace message. This can be the non-zero return code from a command, the prompt message when interactive tracing begins, or the traceback clauses after a syntax error in the program.

>>> Identifies the result of an expression (for TRAC R), the value assigned to a variable during parsir or the value returned from a subroutine call.

>.> Identifies the value "assigned" to a placehold during parsing.

The following prefixes are only used if TRAC Intermediates is in effect:

>C> The name of a variable (after substitution).

>F> The result of a function call.

>L> A literal.

>O> The result of an operation on two terms.

>P> The result of a prefix operation.

>V> The contents of a variable.

Note: *The TRACE instruction should not be confus with the TRACE() built-in function (page 32)*

1.3 System Commands & Subcommand Environments

RXQUEUE [option] [queue_name]

Causes the data which follows it on STDIN (norma the keyboard, but can be the redirected output of command) to be placed in *queue_name* or in the defa REXX data queue SESSION if *queue_name* is omitte

Option may be:

/CLEAR The contents of the currently select REXX data queue is cleared.

/FIFO Data will be placed in the queue in a "first first out" manner. The oldest line in the que will be the first to be retrieved.

/LIFO Data will be placed in the queue in a "last in, first out" manner. The newest line in the queue will be the first to be retrieved.

If the source of the data is the keyboard, the data must be terminated with an end of file character, <Ctrl-z> ('1A'x).

The default queue name can be altered by use of the OS/2 environment variable RXQUEUE.
Example: `SET RXQUEUE=queue_name`

The combined STDOUT and STDERR output can be redirected or "piped" to the a REXX data queue.
Example: `DIR *.XYZ 2>&1 | RXQUEUE`

Note: *The RXQUEUE subcommand should not be confused with the RXQUEUE() built-in function (page 27).*

```
RXSUBCOM DROP     env_name [dll_name]
RXSUBCOM LOAD     env_name [dll_name]
RXSUBCOM QUERY    env_name [dll_name]
RXSUBCOM REGISTER env_name  dll_name entry_point
```

Drops, loads, queries or registers subroutines in *env_name*. *Dll_name* is the name of the Dynamic Link Library module name. *Entry_point* is the name of the function to be executed when called.

Returns 0 if the function completed successfully; otherwise, returns -1 if the parameters are incorrect or:

10 *Env_name* is a duplicate (REGISTER).
30 The *env_name* is currently in use (REGISTER) or does not exist (QUERY, DROP).

SET RXTRACE= ON | OFF

The RXTRACE environment variable can be set to ON prior to starting the REXX interpreter. This will result in the interpreter starting as if the instruction **TRACE '?R'** was the first instruction in the REXX program that is being run. However, if STDERR is redirected when the interpreter is started, RXTRACE=ON has same effect as if **TRACE 'R'** was

the first instruction in the REXX program (i.e. tracin
will not be interactive).

RXTRACE=ON can be reversed with either of th
following command line statements:

```
SET RXTRACE=
SET RXTRACE=OFF
```

1.4 Built-In Functions

OS/2 REXX contains 76 built-in functions that are availabl
without the need to load external function packages (APIs).

ABBREV(full_string, test_string[, length])

Returns 1 if *test_string* is equal to the leading characte
of *full_string* and the length of *test_string* is at least *lengt*
long; otherwise, returns 0.

```
Example:     1 = ABBREV( 'ABCDEF', 'ABD', 2 )
             0 = ABBREV( 'ABCDEF', 'ABD', 3 )
```

ABS(number)

Returns the absolute value of *number* as an unsigne
value formatted according to the current NUMERI
setting. °

```
Example:     128 = ABS( -128 )
             128 = ABS( 128 )
             128 = ABS( 1.28e2 )
```

ADDRESS()

Returns the name of the current environment to whic
host commands are submitted. See the ADDRES
instruction on page 6.

```
Example:     CMD
```

ARG([n[, option]])

Returns the number of arguments, or the *n*th argume
if *n* is specified. If option is E (exists), returns 1 if *nt*
argument exists; otherwise, returns 0. If option is
(omitted), returns 1 if *n*th argument was omitte
otherwise, returns 0.

Note: *Regardless of the number of words, or format*
a command line value passed to a REX
program, this function will always indicate
single argument for the command line string.

B2X(binary_string)

Binary to hexadecimal. Returns the hexadecimal equivalent of *binary_string* (digits 0 and 1). Blanks can be imbedded within *binary_string*, at four-digit boundaries only, for readability. The returned string will use uppercase alphabetic characters for A-F and will not contain any blanks.

```
Example:    C3 = B2X( 1100 0011 )
```

BEEP(frequency, duration)

Sounds the computer's speaker. *Frequency*, in cycles per second (Hertz) with a range of 37 to 32,767, is rounded to the nearest integer. *Duration* is in milliseconds with a range from 1 to 60,000. Functionally equivalent to SOUND() (page 95).

BITAND(string1[, [string2][, pad]])

Returns a string composed of the two input strings *string1* and *string2* logically AND'ed together, bit by bit, with the shorter string optionally padded with *pad*.

```
Example:    '01'b = BITAND( '01'b, '11'b )
            'C3'x = BITAND( 'f7'x, 'c3'x )
```

BITOR(string1[, [string2][, pad]])

Returns a string composed of the two input strings *string1* and *string2* logically OR'ed together, bit by bit, with the shorter string optionally padded with *pad*.

```
Example:    '11'b = BITOR( '01'b, '11'b )
            'F7'x = BITOR( 'f7'x, 'c3'x )
```

BITXOR(string1[, [string2][, pad]])

Returns a string composed of the two input strings *string1* and *string2* logically XOR'ed together, bit by bit, with the shorter string optionally padded with *pad*.

```
Example:    '10'b = BITXOR( '01'b, '11'b )
            '34'x = BITXOR( 'f7'x, 'c3'x )
```

C2D(string[, n])

Character to decimal. Returns the decimal value of the binary representation of *string*. If *n* is specified, *string* is taken as a signed number expressed in *n* characters.

```
Example:    65 = C2D( 'A' )
            97 = C2D( 'a' )
```

C2X(string)

Character to hexadecimal. Converts (unpacks) th character string *string* to its hexadecimal representatio The returned string will use uppercase alphabeti characters for A-F and will not contain any blanks.

```
Example:     '41' = C2X( 'A' )
             '61' = C2X( 'a' )
```

CENTER(string, length[, pad])
CENTRE(string, length[, pad])

Returns a string of *length* with *string* centered in i optionally padded with *pad* (defaults to blank) o truncated as needed. Odd number truncation is applie to the right side.

CHARIN([name][, [start][, length]])

Returns a string up to *length* (default 1) characters rea from the character input stream *name*, optionall beginning at *start* (default 1). Can raise th NOTREADY condition.

Files are implicitly *opened* with the first invocation o CHARIN for the file. Files should be explicitly close with the close option of the STREAM() function (pag 29).

CHAROUT([name], [string][, start])

Returns the count of characters remaining afte attempting to write *string* to the character output strean *name*.

If only *name* is specified, the file *name* is closed.

Start optionally specifies *name*'s write pointer an defaults to the current write position. Can raise th NOTREADY condition.

Files are implicitly *opened* with the first invocation o CHAROUT for the file. Files should be explicitly close with the close option of the STREAM() function (pag 29).

CHARS([name])

Returns the total number of characters remaining in the character input stream *name*; otherwise, returns 0. If *name* is omitted, STDIN is assumed

For streams where the number of characters remaining cannot be determined, returns 1 if any characters remain; otherwise, returns 0. Can raise the NOTREADY condition.

Files are implicitly *opened* with the first invocation of CHARS for the file. Files should be explicitly closed with the close option of the STREAM() function (page 29).

COMPARE(string1, string2[, pad])

Returns 0 if *string1* and *string2* are identical; otherwise, returns the position of the first character that does not match. The shorter string is padded on the right with *pad*, which defaults to a blank, if necessary.

CONDITION([option])

Returns a word from the list shown below (*option* **C**, **I**, or **S**), or a descriptive string (*option* **D**) associated with the current trapped condition indicated by *option*. Option can be **C** (condition), **D** (description), **I** (instruction), or **S** (status). Possible combination of values (one from each column) returned for all *options* but **D** are:

Returns:	If: C	I	S
CALL		•	
DELAY			•
ERROR	•		
FAILURE	•		
HALT	•		
NOVALUE	•		
NOTREADY	•		
OFF			•
ON			•
SIGNAL		•	
SYNTAX	•		

In the instance of this function being issued following a NOVALUE trap for *condition* **D**, the value returned is the uninitialized variable.

COPIES(string, n)

Returns *n* concatenated copies of *string*.

```
Example:     'abcabc' = COPIES( 'abc', 2 )
```

D2C(whole_number[, n])

Decimal to character. Returns a string, with a length a needed, or a length of *n*, containing the ASC representation of *whole_number*. If *n* is specified, represents the character length of the returned string.

```
Example:     'A' = D2C( 65 )
             'a' = D2C( 97 )
```

D2X(whole_number[, n])

Decimal to hexadecimal. Returns a string, with a lengt as needed, or a length of *n*, containing the hexadecim characters equal to *whole_number*. If *n* is specified, represents the character length of the returned strin The returned string will use uppercase alphabet characters for A-F and will not contain any blanks.

```
Example:     '41' = D2X( 65 )
             '61' = D2X( 97 )
```

DATATYPE(string[, type])

If only *string* is specified, the returned result is 'NUM' *string* is a valid REXX number otherwise 'CHAR' returned.

Returns 1 if *string* is of the type *type*; otherwise, returr 0. *Type* can be:

A	alphanumeric - a-z, A-Z, 0-9
B	binary - 0 & 1
C	mixed SBCS (single-byte character set) / DBC (double-byte character set)
D	pure DBCS
L	lowercase - a-z
M	mixed case - a-z & A-Z
N	valid REXX number
S	any character which is permitted in a REXX symbc (page 2)
U	uppercase - A-Z
W	REXX whole number according to DIGITS settin
X	hexadecimal - a-f, A-F, 0-9 & blank

DATE([option])

Returns the local date in the format: dd Mon yyyy°, or in the format specified by *option*:

B	(Basedate) *dddddd* - days since & including Jan. 1, 0001. °
C	(Days this century) *ddddd*. (Personal REXX only)
D	(Days this year) *ddd*. °
E	(European) *dd/mm/yy*.
J	(Julian) *yyddd*. (Personal REXX only)
L	(Local or implementation defined) *dd Month yyyy*.
M	(Month) Full English name of the current month.
N	(Normal) default format. *dd Mon yyyy* °
O	(Ordered) *yy/mm/dd*.
S	(Sorted) *yyyymmdd*.
U	(USA) *mm/dd/yy*.
W	(Weekday) Weekday name in English in mixed case (first letter uppercase).

DELSTR(string, n[, length])

Deletes the sub-string of *string* that begins at the *n*th character and is of length *length*. If *length* is not specified, the rest of the string is deleted (including the *n*th character).

```
Example:     'abe' = DELSTR( 'abcde, 3, 2 )
```

DELWORD(string, n[, length)

Deletes the sub-string of *string* that begins at the *n*th word. *Length* indicates the number of blank delimited words. If *length* is not specified, the remaining words are deleted.

```
Example:     'ab cd' = DELWORD( 'ab cd ef gh', 3 )
```

DIGITS()

Returns the current setting of NUMERIC DIGITS. °

DIRECTORY([new_directory])

Returns the result of changing the current directory to *new_directory*, if it is specified and *new_directory* exists. DIRECTORY() must be used without *new_directory* if the value of the current directory is needed.

Note: *Use of the DIRECTORY() function is preferred over the external 'CD' command to change the current directory since the external CD command* *requires shelling out to CMD.EXE and, if the REXX program is not running under CMD.EXE (e.g. PMREXX), then the directory change is lost.*

ENDLOCAL()

Restores the drive, directory and environment variables in effect before the last SETLOCAL function. Returns 1 if environment successfully restored; otherwise, returns 0.

ERRORTEXT(n)

Returns the text of the error message associated with error number *n* (0 ≤ n ≤ 99). Undefined error numbers return a null string. Appendix D (page 215) lists all of the error messages and their meaning.

```
Example:    'File Table full' = ERRORTEXT( 1 )
```

FILESPEC(option, filespec)

Parses the complete file name in *filespec* and returns the selected element of *filespec* as indicated by *option* or returns a null string if the requested token is not found. *Filespec* need not be an existing file. Similar in function to PARSEFN (page 87). Example of returned string given the file system name of *C:\OS2\DLL\REXX.DLL* for *option*:

```
D   (Drive)  'C:'
P   (Path)   '\OS2\DLL\'
N   (Name)   'REXX.DLL'
```

FORM()

Returns the current setting of NUMERIC FORM (i.e. <u>SCIENTIFIC</u> or ENGINEERING).

FORMAT(number[, [before][, [after] →
→ [,expp][,expt]]]])

Returns *number* rounded and formatted with *before* and *after* specifying the size of the integer and fraction parts respectively. *Expp* specifies the number of places for the exponent and *expt* specifies the trigger point for the use of exponential notation. If *before* is not large enough to contain the integer part of *number* an error results.

```
Example: with x = 123.456
    123.45 = FORMAT( x, 3, 2 )
    123E+2 = FORMAT( x, 3, 2, 1, 1 )
```

FUZZ()

Returns the current setting of NUMERIC FUZZ (default is 0). °

INSERT(new, target[, [n][, [length][, pad]]])

Inserts the string *new*, padded to *length* with *pad*, into the string *target* after the *n*th character. Default value of *n* is zero and default value of *pad* is blank. Use of the defaults is equivalent to specifying *new* || *target*.

```
Example:
    'ab?cd' = INSERT( '?', 'abcd', 2 )
```

LASTPOS(needle, haystack[, start])

Returns the character position of the last occurrence of *needle* in *haystack*; otherwise, returns 0. The search is started at the last character of *haystack*, or *start*, and proceeds right to left.

```
Example:      5 = LASTPOS( 1, 121314 )
```

LEFT(string, length[, pad])

Returns a string of length *length* containing the left-most *length* characters of *string*, padded with *pad* or truncated on the right as needed.

```
Example:      'abc' = LEFT( 'abcdef', 3 )
```

LENGTH(string)

Returns the length of *string*. °

LINEIN([name][, line][, count]])

Returns *count* line(s) read from the character input stream *name*. *Count* can be either 1 - read one line, or 0 - reposition the read pointer. *Line* (which must be 1 if used) specifies the line position in the stream from which to read. Can raise the NOTREADY condition.

Files are implicitly *opened* with the first invocation of LINEIN for the file. Files should be explicitly closed with the close option of the STREAM() function (page 29).

LINEOUT([name][, string][, line]])

Returns the count of lines remaining (0 - none, 1 - otherwise) after attempting to write *string* to the character output stream *name*. *Line* (which must be 1 if used) specifies the line position in the stream at which to write. Can raise the NOTREADY condition.

If only name is specified, the file *name* is closed.

Files are implicitly *opened* with the first invocation of LINEOUT for the file. Files should be explicitly closed with the close option of the STREAM() function (page 29).

LINES([name])

Returns 1 if any lines remain in the character input stream *name*; otherwise, returns 0. Can raise the NOTREADY condition.

Files are implicitly *opened* with the first invocation o LINES for the file. Files should be explicitly closed with the close option of the STREAM() function (page 29).

MAX(number[, number] ...)

Returns the largest *number* out of the list specified. °

MIN(number[, number] ...)

Returns the smallest *number* out of the list specified. °

OVERLAY(new, target[, [n][, length][, pad]]])

Returns the string *target*, which, starting at the *n*th character, is overlaid with the string *new*, padded with *pad* or truncated to *length*.

| Example: 'ab?de' = OVERLAY('?', 'abcde', 3)

POS(needle, haystack[, start])

| Returns the character position of the first occurrence of *needle* in *haystack*; otherwise, returns 0. The search is started at the first character of *haystack*, or *start*, and proceeds left to right. °

| Example: 2 = POS(2, 121212)

QUEUED()

Returns the number of lines remaining in the currently active REXX data queue.° The name of the currently active REXX data queue can be interrogated or set with the RXQUEUE() function (page 27).

RANDOM(max)
RANDOM([min][, max][, seed])

Returns a quasi-random, non-negative, whole number in the range 0 to 999, or *min* to *max* inclusive. *Max* minus *min* must not exceed 100,000. A whole number, *seed*, can be specified if repeatable results are desired. °

REVERSE(string)

Returns *string*, swapped end for end.

```
Example:     'edcba' = REVERSE( 'abcde' )
```

RIGHT(string, length[, pad])

Returns a string of length *length* containing the right-most *length* characters of *string*, padded with *pad* or truncated on the left as needed.

```
Example:     'de' = RIGHT( 'abcde', 2 )
```

RXFUNCADD(function_name, module, entry_name)

Returns 0 if the external function *function_name* located in *module* with an entry point name *entry_name* is successfully registered (added as being available); otherwise, returns an error code identifying the error.

A return code of 0 does not indicate that the module (i.e. DLL) and *entry_name* actually exist.

Note: *Entry_name is case sensitive and both function_name and entry_name should be specified as either literals or assigned variables.*

RXFUNCDROP(function_name)

Returns 0 if *function_name* is successfully removed (unregistered); otherwise, returns 1. When *function_name* is removed, it becomes unavailable to this session and all other sessions until a subsequent RxFuncAdd() restores its availability. Thus, the RXFUNCDROP function should be avoided in normal operation.

RXFUNCQUERY(function_name)

Returns 0 if *function_name* is registered (available for use); otherwise, returns 1.

RXMESSAGEBOX(text[, title][, button][, icon])

Returns a whole number resulting from a button which was selected in a text box created with this function call and containing *text*, with the optional *title*.

This function is only available to a REXX program running under PMREXX or called from a Presentation Manager application. It can also be used with a program launched from the command line with the following:

```
START /PM CMD /C program_name [parameters]
```

An optional *button* and *icon* can also be included in the text box. The style of *button*, which defaults to OK, includes:

OK
: A single OK button.

OKCANCEL
: Both an OK button and a Cancel button.

CANCEL
: A single Cancel button.

ENTER
: A single Enter button.

ENTERCANCEL
: An Enter button and a Cancel button.

RETRYCANCEL
: A Retry button and a Cancel button.

ABORTRETRYCANCEL
: An Abort button, a Retry button and a Cancel button.

YESNO
: A Yes button and a No button.

YESNOCANCEL
: A Yes button, a No button and a Cancel button.

Icon designates an optional icon to be displayed in the text box. *Icon* can be:

NONE
: No icon is displayed.

HAND
The hand icon is displayed.

QUESTION
A question mark icon is displayed.

EXCLAMATION
An exclamation icon is displayed.

ASTERISK
An asterisk icon is displayed.

INFORMATION
The information icon, identical to the icon shown on the desktop, is displayed.

QUERY
The query icon is displayed.

The whole number returned, as a result of the button being selected, will be:

1	OK button	5	Ignore button
2	Cancel button	6	Yes button
3	Abort button	7	No button
4	Retry button	8	Enter key

Note: *The dialogue box with the text "The REXX procedure has ended." which appears at the completion of all programs run with PMREXX can not be suppressed. It is compiled into the PMREXX program.*

RXQUEUE(CREATE[, queue_name])
RXQUEUE(DELETE, queue_name)
RXQUEUE(GET)
RXQUEUE(SET, new_queue_name)

Create returns the name of the newly created REXX data queue - either *queue_name* or a system generated name if *queue_name* is omitted or a queue named *queue_name* already exists.

Delete returns the following after attempting to delete *queue_name*:

0	Que has been deleted.
5	Not a valid queue name.
9	Queue name does not exist.
10	Queue is busy; wait is active.
12	A memory failure occurred.
1000	Initialization error; check OS2.INI

Get returns the name of the queue currently in use ir this session.

Set returns the name of the queue currently in use ir this session and replaces the current queue witl *new_queue_name*.

The RXQUEUE() function should not be confused witl the RXQUEUE subcommand (RXQUEUE.EXE - page 14)

SETLOCAL()

Returns 1 if the current working drive and directory, anc the current values of the OS/2 environment, are successfully saved by the SETLOCAL function otherwise, returns 0.

SIGN(number)

Returns a number that indicates the sign of *number* ('-1' '0', or '1').

SOURCELINE([n])

Returns the *n*th line of the program, or the number o lines in the program if *n* is omitted. Returns 0 if *n* i: omitted, or a null string if *n* is specified when the source program is not available (e.g. the program is being rui from the macrospace).

SPACE(string[, [n][, pad]])

Returns a formatted string of the blank-delimited word: in *string* with *n pad* characters between each word.

`Example:  'abc def ghi' = SPACE( 'abc  def   ghi'`

STREAM(file_name[, operation[, stream_command]])

Returns information describing the state of the character stream *file_name* or the result of an operation on *file_name*. Operation can be:

C	(Command)
D	(Description)
S	(State)

If *operation* is **C**, *stream_command* can be:

OPEN {READ | WRITE}

Opens *file_name* for both reading and writing unless READ or WRITE is specified (WRITE implies READ access). If neither is specified, both are implied.

CLOSE

Closes *file_name*. Returns "READY:" if close is successful otherwise an appropriate error indication. Returns a null string if *file_name* was not previously OPENed. This option is the preferred method for closing a file vs. using one of the I/O functions with just the *file_name* parameter.

SEEK {= | < | + | -} offset

Sets the read / write position of the previously OPENed *file_name* to a value specified by *offset*. *Offset* can only represent a number of characters. Therefore, the position of a line within a file is implied only if the file contains fixed-length records thus allowing the number of characters to be calculated.

Returns the new position in the file if the operation was successful; otherwise, returns an error indicator.

The *offset* number can be preceded by:

=	Explicitly specifies the *offset* from the beginning of the stream.
<	Specifies the *offset* from the end of the stream.

+	Specifies the *offset* forward from the current position.
-	Specifies the *offset* backward from the current position.

QUERY [EXISTS | SIZE | DATETIME]

EXISTS Returns the full path of *file_name* or a null string if *file_name* does not exist.

`Example: D:\os2\dll\rexx.dll`

SIZE Returns the size, in bytes, of *file_name*.°

`Example: 248352`

DATETIME Returns the date and time stamps of *file_name*.

`Example: 09-29-93  17:46:42`

If operation is **C**, STREAM returns READY: if *stream_command* is successful; otherwise, returns NOTREADY: concatenated with a numeric representation of the reason for the condition.

`Example: NOTREADY:110`

If *operation* is **S**, STREAM returns ERROR, NOTREADY, READY or UNKNOWN.

`Example: UNKNOWN`

If *operation* is **D**, the results returned are identical to the State operation except that the returned string is followed by a colon and, if available, additional information about ERROR or NOTREADY states.

`Example: UNKNOWN:`

Note: The STREAM() function will not "see" hidden or system files. The SysFileTree() function in REXXUTIL (page 45) must be used if checking for a file that has the hidden or system attribute set.

STRIP(string[, [option][, char]])

Returns a string with leading, trailing, or both leading and trailing *char* characters removed from *string* when the first character of *option* is **L**, **T** or **B** respectively. *Char* can only be one character long and defaults to blank.

```
Example:     'abc' = STRIP( '.abc.', 'B', '.' )
```

SUBSTR(string, n[, [length][, pad]])

Returns the portion of *string* that begins at the *n*th character and is of length *length*, padded with *pad* if necessary.

```
Example:     'bcd' = SUBSTR( 'abcdef', 2, 3 )
```

SUBWORD(string, n[, length])

Returns the portion of *string* that starts with the *n*th word and is up to *length* blank-delimited words long.

```
Example:  'cd ef' = SUBWORD( 'ab cd ef gh', 2, 2 )
```

SYMBOL(name)

Returns 'BAD' if *name* is not a valid REXX symbol, 'VAR' if it is the name of a variable, or 'LIT' (literal) otherwise.

TIME([option])

Returns the local time in the 24-hour format: **hh:mm:ss**, or in the format specified by *option*:

C	(Civil) *hh:mmxx* (where xx is 'am' or 'pm') ° `Example: 1:12pm`
E	(Elapsed seconds.hundredths) *0* (first invocation) or *ssssssss.uu0000* °
H	(Hours since midnight) *hh* °
L	(Long) *hh:mm:ss.uu0000* °
M	(Minutes since midnight) *mmmm* °
N	(Normal) default format *hh:mm:ss*
R	(Reset) *sssssss.uu0000* (and resets elapsed time) °
S	(Seconds since midnight) *sssss* °

TRACE([option])

Returns trace actions currently in effect. If *option* is specified, it must be the valid prefix (?), one of the valid alphabetic character options (A, C, E, F, I, L, N, O, or R) associated with the TRACE instruction, or both. Unlike the TRACE instruction, the TRACE function alters the trace action even if interactive debugging is active.

See page 13 for a description of TRACE output.

TRANSLATE(string[, [out_table][, [in_table] →
→ [, pad]]])

Returns a string with characters in *string* that are in *in_table* translated to the corresponding character in *out_table* (*out_table* is padded with *pad* if needed). If neither translate table is specified, *string* is translated to uppercase.

```
Example:  '12:34' = TRANSLATE( '12/34', ':', '/' )
```

TRUNC(number[, n])

Returns the integer part of *number* and *n* decimal places.
o

VALUE(name[, newvalue[, selector]])

Returns the value of the symbol *name*. A new value can be supplied in *newvalue*.

Selector can be 'OS2ENVIRONMENT', or an expression that evaluates to it, to access or set system environment variables. System environment variables altered with this function remain in effect for the current program only. Functionally equivalent to DOSENV() (page 73) when *selector* is OS2ENVIRONMENT.

Note 01: Use of the VALUE() function is preferred over the external 'SET' command to change an environment variable since the external SET command requires shelling out to CMD.EXE and, if the REXX program is not running under CMD.EXE (e.g. PMREXX), then the environment variable change is lost.

Note 02: There is no way to change the global environment.

VERIFY(string, characters[, [option][, start]])

Returns the position of the first character in *string* that either: is not, or is, (depending on *option*) in *characters*; otherwise, returns 0. *Option* can be either <u>NOMATCH</u> or MATCH (or just the first character of either). *Start*, which defaults to 1, indicates the position in *string* where the search begins.

```
Example:     4 = VERIFY( '123,456.78', '.,', 'M' )
             1 = VERIFY( '123,456.78', '.,', 'N' )
```

WORD(string, n)

Returns the *n*th blank-delimited word in *string*. Returns a null string if there are fewer than *n* words in *string*.

```
Example:     'cd' = WORD( 'ab cd ef gh', 2 )
```

WORDINDEX(string, n)

Returns the character position of the *n*th blank-delimited word in *string*. Returns 0 if there are fewer than *n* words in *string*. °

```
Example:     9 = WORDINDEX( 'abc def ghi', 3 )
```

WORDLENGTH(string, n)

Returns the length of the *n*th delimited word in *string*. Returns 0 if there are fewer than *n* words in *string*. °

```
Example:     2 = WORDLENGTH( 'a bc def', 2 )
```

WORDPOS(phrase, string)

Returns the word number of the first blank-delimited word of *phrase* in *string*; otherwise, returns 0. °

```
Example:     3 = WORDPOS( 'ef', 'ab cd ef gh' )
```

WORDS(string)

Returns the number of blank-delimited words in *string*. Returns 0 if *string* has a length of zero or contains only blanks. °

```
Example:     4 = WORDS( 'ab cd ef gh' )
```

XRANGE([start][, end])

Returns a string of all one-byte codes between and including the values *start* (<u>'00'X</u>) and *end* (<u>'FF'X</u>).

```
Example:     'ABCDEFGHI' = XRANGE( 'A', 'I' )
```

X2B(hex_string)

Hexadecimal to binary. Returns a binary string equivalent to *hex_string* (a string of hexadecimal characters which can contain an imbedded blank at byte boundaries for readability)). The returned string will have a length that is a multiple of four and will contain only the characters 0 and 1.

```
Example:     '10101011' = X2B( 'ab')
```

X2C(hex_string)

Hexadecimal to character. Returns (packs) a character string equivalent to *hex_string* (a string of hexadecimal characters which can contain an imbedded blank at byte boundaries for readability).

```
Example:     'A' = X2C( '41' )
```

X2D(hex_string[, n])

Hexadecimal to decimal. Returns a whole number equivalent to *hex_string* (a string of hexadecimal characters which can contain an imbedded blank at byte boundaries for readability). °

```
Example:     65 = X2D( '41' )
```

This page intentionally blank

2. REXX External Function Modules

Many external function packages (APIs - Application Program Interface modules) are available for REXX running under OS/2. REXXUTIL is distributed as an integral part of OS/2. A different version of REXXUTIL is used with Object REXX. Other APIs can be obtained from commercial software sources, Internet sites, electronic bulletin boards (BBS), etc.

The external function module containing each of the following functions is indicated by the module name at the top of the respective page. Where a function was introduced with a particular level of the module, or pertains to a particular version of the module, an abbreviation appears at the end of the function definition line. These abbreviations, and the external function packages they refer to, are:

```
REXXUTIL (Classic REXX)
REXXUTIL (Object REXX only)                    (OBJ)
REXXLIB  (Quercus Systems)
RXWINDOW (Quercus Systems)
```

2.1 REXXUTIL Functions

REXXUTIL is an external function package included as part of REXX and OS/2. Two separate versions of REXXUTIL are available beginning with Warp Version 4 - CREXUTIL.DLL and OREXUTIL.DLL. Depending on the currently selected version of REXX (Classic vs. Object), its respective DLL is renamed to REXXUTIL.DLL.

The currently selected version of REXX can be toggled to the alternate version on Warp Version 4 via the \OS2\SWITCHRX.CMD file. Each time the system is switched from Classic REXX to Object REXX, it is necessary to run \OS2\WPSINST.CMD after re-IPLing the system to register the classes used by Object REXX.

The REXXUTIL API distributed with Object REXX contains additional functions not available in the REXXUTIL API provided with Classic REXX. These additional functions cannot be used with Classic REXX. REXXUTIL.DLL provides functions which deal with the following:

```
OS/2 System Commands
User text or screen I/O
OS/2 INI file I/O
File System functions
Mutex & Event Semaphores                          (OBJ)
National Language functions                       (OBJ)
Macrospace functions                              (OBJ)
Function Library routines                         (OBJ)
```

REXXUTIL.DLL must be added to the REXX processor using the built-in function *RxFuncAdd*. This can be accomplished by making STARTUP.CMD a REXX command file (by making its first line a REXX style comment - /* */) and including:

```
call RxFuncAdd 'SysLoadFuncs','RexxUtil','SysLoadFuncs'
call SysLoadFuncs
```

Once the REXXUTIL functions or any other external functions are loaded, they are useable by all OS/2 sessions.

All possible effort has been made to include all return codes which can result from these functions; however, there is no assurance that those return values listed for each function represent all of the possible return codes that exist.

New functions that have been added to REXXUTIL since the initial release of OS/2 2.1 are indicated by the release level when the function was first introduced being shown at the end of the line containing the function's parameters.

Note: *The REXXUTIL text window functions will not function properly in PMREXX or other Presentation Manger (GUI) environments.*

SysAddFileHandle(number) **(OBJ)**

Returns the number of file handles available to the current session after adding *number* to the previously available number of file handles and making the sum file handles available to the session. If *number* is zero, the current number of file handles is returned. Similar in function to DOSFILEHANDLES (page 73).

Note: *File handles are used every time a file is accessed with REXX I/O functions such as CHARIN / CHAROUT, LINEIN / LINEOUT, CHARS, LINES, and STREAM. Other services invoked* *indirectly from REXX (e.g. SysGetMessage() or other API routines) may also create open file handles. At least 3 handles are usually in use for the standard input stream, standard output. stream, and standard error stream.*

SysAddRexxMacro(function_name, file_name → → [, position]) (OBJ)

Returns the RexxAddMacro() return code after adding *file_name* to the REXX macrospace as *function_name*. *Function_name* is the name used to call the function. If no file extension is specified for *file_name*, .CMD is assumed. When a path is not specified for *file_name*, the current directory and PATH environment are searched. *Position* is either **B**, indicating that the macrospace should be searched before the list of registered functions and functions on disk; or, **A** indicating that the macrospace should be searched after a search is made of registered functions and functions on disk.

0	Macro added successfully
1	Error - no storage available
7	Error - source not found
8	Error - invalid position

SysBootDrive() (OBJ)

Returns the one character disk letter, in uppercase, followed by a colon of the system boot drive. Similar in function to DOSBOOTDRIVE (page 67).

SysClearRexxMacroSpace() (OBJ)

Returns the RexxClearMacroSpace() return code after removing all macros from the macrospace.

0	Macrospace cleared
2	Error - macrospace already cleared

SysCloseEventSem(handle) **(OBJ)**

Returns the DosCloseEventSem return code after closing the event semaphore identified by *handle* and releasing its associated storage. *Handle* is the value returned by a successful call to SysCreateEventSem().

0	Semaphore closed
6	Error - invalid handle
301	Error - semaphore busy

SysCloseMutexSem(handle) **(OBJ)**

Returns the DosCloseMutexSem return code after closing the mutex semaphore identified by *handle* and releasing its associated storage. *Handle* is the value returned by a successful call to SysCreateMutexSem().

0	Semaphore closed
6	Error - invalid handle
301	Error - semaphore busy

SysCls()

Returns 0 and clears a window but not necessarily the entire screen. Similar in function to SCRCLEAR (page 92).

SysCopyObject(object_name, object_destination) **V3**

Returns 1 if *object_name* was successfully copied to *object_destination*; otherwise, returns 0. If the object already exists in the destination location, it is not copied and a 0 is returned.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

Note 01: The copied object will not have an OBJECTID whether the original object had one assigned or not.

Note 02: Some of the object's other properties are not copied along with the object. Specifically, ASSOCTYPE= belonging to the original object *does not appear on the copy. This is consisten with what occurs when using drag & drop to copy an object.*

SysCreateEventSem([name]) **(OBJ)**

Returns an event semaphore handle after attempting tc create or open an OS/2 event semaphore; otherwise returns a null string. The handle that is returned can be used with SysOpenEventSem(), SysCloseEventSem() SysResetEventSem(), SysPostEventSem(), and SysWaitEventSem(). SysCreateEventSem() returns a nul string if the event semaphore cannot be created or opened.

If you omit the optional event semaphore *name* SysCreateEventSem() creates an unnamed, shared even semaphore. If you specify *name*, SysCreateEventSem(opens the semaphore if the semaphore has already been created. The file system validates semaphore names which must include the prefix '\SEM32\'.

SysCreateMutexSem([name]) **(OBJ)**

Returns a mutex semaphore handle after attempting tc create or open an OS/2 mutex semaphore; otherwise returns a null string. The handle that is returned can be used with SysOpenMutexSem(), SysCloseMutexSem() SysRequestMutexSem(), and SysReleaseMutexSem() SysCreateMutexSem() returns a null string if the mute semaphore cannot be created or opened.

If you omit the optional mutex semaphore *name* SysCreateMutexSem() creates an unnamed, shared mutex semaphore. If you specify *name* SysCreateMutexSem() opens the mutex semaphore if the semaphore has already been created. The file system validates mutex semaphore names, which must include the prefix '\SEM32\'.

```
SysCreateObject( class_name, title, location →
                          → [, setup_string][, option] )
```

Returns 1 if a new object class was created; otherwise returns 0.

Class_name is the name of the WPS class of which the object is a member and *title* is the new object's title. A new line character, '0A'x, can be included in *title*. The occurrence of the escape character ^ ('5E'x) also causes a new line to be created; however, 2nd and subsequent escape characters used for this purpose appear to be ignored.

Location can be either an object ID (any unique string preceded with a '<' and terminated with a '>') or a full file system path.

Setup_string optionally must contain a WinCreateObject string which is comprised of a series of "key name=value" pairs that change the behavior of the object. "Key names" are separated by semicolons and "values" are separated by commas.

Note: *If a value includes a semicolon (; - '3B'x) or a comma (, - '2C'x), it must be "escaped" by preceding it with a caret (^ - '5E'x).*

Option is a string which indicates the action to be taken if the object class already exists and can be: FAIL, REPLACE or UPDATE.

Section 4.2 describes the predefined *class_names* beginning on page 124 and section 4.3 contains the *setup_string* data beginning on page 126.

SysCreateShadow(object_name, object_destination)

V3

Returns 1 if a shadow of *object_name* was successfully created at the specified location, *object_destination*; otherwise, returns 0.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

SysCurPos([row, col])

Returns the current relative cursor position on the screen as two whole numbers, row and column, and optionally moves the cursor to the position specified in *row* and *col*.° The first column of the first row is returned as 0 0. Similar in function to CURSOR (page 65).

```
Example: 4 1
```

SysCurState({ON | OFF})

Returns a value of 0 and changes the current state of the cursor to displayable (*ON*) or non-displayable (*OFF*).

SysDeregisterObjectClass(class_name)

Returns the resultant code from WinDeregisterObjectClass: 1 if *class_name* was successfully unregistered; otherwise, returns 0. Section 4.2 describes the predefined WPS classes beginning on page 124.

SysDestroyObject(object_name)

Returns the resultant code from WinDestroyObject: 1 if *object_name* is destroyed; otherwise, returns 0. *Object_name* is either the object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created or the full file system name of the object to be destroyed. Some of the predefined object IDs are shown in Section 4 beginning on page 107.

SysDriveInfo(drive)

Returns 4 words describing *drive* or a null string if *drive* contains an expression which does not evaluate to a drive letter or if the specified drive is not ready.

Word 1 contains the *drive* letter, in uppercase, followed by a colon. Word 2 contains the number of free bytes on *drive*. Word 3 contains the total number of bytes on *drive*. Word 4 contains the volume label from *drive*. °

```
Example:     D:  19118080  63031296   OS2
```

SysDriveMap([drive], [opt])

Returns a string of all accessible drives as the drive letter in uppercase followed by a colon, optionally beginning with *drive*. If *drive* is omitted, the default is C. *Opt* causes only the specified type of drives to be returned and can be:

<u>USED</u>	Only drives which are accessible or in use.
FREE	Drives which are free or not in use (i.e. all drive letters beyond the last drive used).
LOCAL	Only local drives are returned.
REMOTE	Only LAN and IFS attached drives are returned.
DETACHED	Drives which are detached LAN resources.

```
Example:    C:  D:  E:
```

SysDropFuncs()

This function is an entry point within REXXUTIL that can be invoked or called to drop all of the REXXUTIL functions. It returns a null string if REXXUTIL was loaded but will trap on SYNTAX if REXXUTIL is not loaded.

SysDropLibrary(DLL[, drop_routine]) **(OBJ)**

Returns 0 after successfully calling the *drop_routine* in *DLL* to terminate and deregister all of the functions in *DLL*; otherwise, returns 1. If *drop_routine* is not specified, SysDropLibrary() will call ordinal routine #2 in the DLL.

SysDropRexxMacro(name) **(OBJ)**

Returns the return code from RexxDropMacro after removing *name* from the macrospace.

0	Macro removed
2	Error - *name* not found

SysElapsedTime([option]) **(OBJ)**

Returns a time value in the format *sssssssss.uuuuuu.*° The fractional part will always have six digits.

Option can be:

E	Elapsed - returns the number of seconds and microseconds since the elapsed time clock was started or reset. (default)
R	Reset - returns the number of seconds and microseconds since the elapsed time clock was started or reset and also simultaneously resets the elapsed time clock to zero.

Note: *SysElapsedTime() reports an elapsed time using the OS/2 high-frequency timer services. The high-frequency timer services have a higher timer resolution than the timer services used by the TIME() built-in function. SysElapsedTime() maintains a single, process-wide time stamp for measuring elapsed time. In contrast with the TIME() built-in function, the time stamp is not saved and restored on subroutine calls and multiple calls to SysElapsedTime() in a single instruction will use different versions of the time stamp.*

SysFileDelete(file_name)

Returns one of the following codes after attempting to delete *file_name*. Functionally equivalent to DOSDEL() (page 69).

0	File deleted successfully.
2	Error - file not found.
3	Error - path not found.
5	Error - access denied.
26	Error - Not a DOS disk.
32	Error - sharing violation.
36	Error - Sharing buffer exceeded.
87	Error - invalid parameter.
123	Error - invalid file name.
206	Error - file name exceeds range.

SysFileSearch(needle, haystack, stem, [opt])

Returns a whole number indicating the result of searching for the string *needle* in the file *haystack*. *Stem* is a variable where all lines found in *haystack* which contain *needle* are placed. *Stem*.0 will contain the total number of lines found if the return code from the function is 0.

Opt can optionally contain the letters **C** (case sensitive search) and/or **N** (include line numbers on matches). If line numbers are requested via the **N** option, the line number is the first word in the string placed in the stem element. °

0	Successful.
2	Error - not enough memory.
3	Error opening file.

SysFileSystemType(drive) **(OBJ)**

Returns the name of the file system used on *drive* or a null string if *drive* is not accessible. *Drive* can be specified with a drive letter, optionally followed by a colon. Functionally equivalent to DOSFILESYS() (page 74). The values returned can be:

null	Drive not accessible
CDFS	CD-ROM file system
FAT	FAT file system
HPFS	High performance file system
LAN	Network drive

SysFileTree(filespec, stem[, opt] →
→ [, targ_attr][, new_attr])

Returns a whole number indicating the success (0) or failure (1 - not enough memory) of a file search which finds all files which match *filespec*.

Stem will contain the file characteristics (date, time, size, attribute string and fully qualified file name) for matching entries.

Note: *Since file names on an HPFS drive may include spaces, the following technique is recommended for parsing the data in stem:*

```
PARSE VALUE stem.n WITH,
    file_date,
    file_time,
    file_size,
    file_attributes,
    file_path_and_name
file_path_and_name = STRIP(file_path_and_name)
```

Opt contains any logical combination of the of the following:

B	Search for both files and directories. (default)
D	Search for directories only.
F	Search for files only.
S	Scan subdirectories recursively. (non-default)
T	Return time and date fields concatenated in the form YY/MM/DD/HH/MM.
O	Only report the fully qualified file name rather than the default of: MM/DD/YY° HH:MM$^{a}_{p}$° size° attr_list name

The attribute lists returned or specified are 5 byte positional character strings containing the letters Archive, Directory, Hidden, Read-only, or System respectively or another character representing that the particular attribute is not set (returned list) or as indicated below for *targ_attr* and *new_attr*.

The target attribute list, *targ_attr*, is used as a mask when searching for *filespec* matches. Only *filespecs* which match the mask will be reported. The default mask is '*****' and each position corresponds to ADHRS noted above. Each position of the *targ_attr* mask can contain:

*	The specified attribute can be set or clear.
+	The specified attribute must be set.
-	The specified attribute must be clear.

The new attribute list, *new_attr*, is used as a mask to set the attributes of all files matching *targ_attr*. The default mask is '*****' and each position corresponds to ADHRS noted above. Each position of the *new_attr* mask can contain:

*	The specified attribute will not be changed.
+	The specified attribute will be set.
-	The specified attribute will be cleared.

Note: *new_attr will have no effect when option 'O' is specified.*

SysGetCollate([country], [code_page]) **(OBJ)**

Returns the 256-byte collating table *country* and optionally *code_page* combination. If *country* or *code_page* are omitted or contain 0 (the default), the collating table for the current system country code and/or current process code page is returned.

SysGetEA(file, ea_name, ea_value)

Returns 0 indicating the *ea_name* extended attribute for *file* has been retrieved and placed in *ea_value*; non-zero otherwise. All of the standard system extended attributes are null-terminated strings. The STRIP() function can be used to remove the terminating null character.

```
Example:  ea_value = STRIP( ea_value, 'T', '00'x )
```

SysGetKey(<u>ECHO</u> | NOECHO)

Returns the next key from the keyboard buffer without waiting for the Enter key. The key is echoed or not according to the option specified. A list of all of the possible combinations of keys and the data returned by this function for each key is listed in Appendix B (page 211). Functionally equivalent to INKEY() (page 83).

SysGetMessage(num[, file[, string_1] →
→ ... [, string_9])

Returns the message associated with *num* in the message file *file*. *String_n* contains replacement fields in the message designated by %n, in the range %1 - %9.

```
Example:
   say SysGetMessage( 46, 'OSO001.MSG', 'Test' )
```
yields:
```
    SYS0046: The Test printer is out of paper.
```

```
SysIni( [inifile], app, key, value )              1
SysIni( [inifile], app, key )                     2
SysIni( [inifile], app, key, 'DELETE:' )          3
SysIni( [inifile], app, 'DELETE:' )               4
SysIni( [inifile], app, 'ALL:', stem )            5
SysIni( [inifile], 'ALL:', stem )                 6
```

Returns ERROR: if the function invocation results in an error condition.

Returns a null string after successfully setting *key* to *value* for *app* (form 1).

Returns the value associated with *key* for *app* (form 2).

Returns a null string after successfully deleting *key* (form 3) or *app* and all of its associated *key*s (form 4).

Returns a null string after placing all *key* values for *app* in *stem* (form 5).

Returns a null string after placing all application names or other meaningful names in *stem* (form 6).

Inifile is the full file system name of an .INI file or USER (value of the SET USER_INI= environment variable), SYSTEM (value of the SET SYSTEM_INI= environment variable) or BOTH. If a user .INI file is specified, a setting function (form 1) will cause the user .INI file to be created if it does not already exist or to be updated if it does exist.

App is an application, or other meaningful value, that key word data should be saved with. *Key* is the name of the keyword associated with *app*.

Value is a string associated with *key* for a specific *app*.

Stem is the name of a stem variable used to store the resultant information with *stem*.0 containing the number of elements in *stem*.

See the SysIni() function in the OS/2 *REXX Information* on-line help facility for an example of how to list all of the program objects in your system.

SysLoadFuncs() **(OBJ)**

Returns a null string after loading all of the functions contained in REXXUTIL.DLL.

SysLoadLibrary(DLL[, load_routine]) **(OBJ)**

Returns 0 if *load_routine* was successful; otherwise, 1. *DLL* is the name of the library module. *Load_routine* is the name of the function loader routine in the *DLL*. If *load_routine* is not specified, SysLoadLibrary() will call ordinal routine #1 in the *DLL*.

SysLoadRexxMacroSpace(file) **(OBJ)**

Returns the RexxLoadMacroSpace return code after loading functions from a saved macrospace file previously created with SysSaveMacroSpace().

0	Functions loaded successfully.
1	Error - no storage available
2	Error - macro not found
4	Error - macro already exists
5	Error - file error
6	Error - signature error

SysMapCase(string[, country,] [code_page]) **(OBJ)**

Returns *string* uppercased according to the *country* and *code_page* specified. A *country* value of 0 results in a translation table for the current system country code. A *code_page* value of 0 results in a translation table for the current process code page.

SysMkDir(dirspec)

Returns one of the following codes resulting from the attempt to create directory *dirspec*. Functionally equivalent to DOSMKDIR() (page 75).

0	Directory created successfully.
2	Error - file not found.
3	Error - path not found.
5	Error - access denied.
26	Error - Not a DOS disk.
87	Error - invalid parameter.
108	Error - drive locked.
206	Error - filename exceeds range.

SysMoveObject(object_name, object_destination) **V3**

Returns 1 if *object_name* was successfully moved to *object_destination*; otherwise, returns 0. If the object already exists in the destination location, it is not moved and a 0 is returned.

Both *object_name* and *object_destination* can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

SysNationalLanguageCompare(string1, string2 →
→ [, country][, code_page]) **OB**

Returns a value indicating the result of comparing *string* and *string2* using a country-specific collating table. A *country* value of 0 results in a collating table for th current system country code. A *code_page* value of results in a collating table for the current process cod page.

0	The two strings are equal.
1	The first string is longer than the second string.
-1	The second string is longer than the first string.

Note: *Comparisons are done using the Rexx strin comparison rules. The strings are compared fo the length of the shorter string. If the leading par is equal, then the longer string is considered to b larger. An equal comparison can only occu between strings of equal lengths.*

SysOpenEventSem(handle) **(OBJ**

Returns the DosOpenEventSem return code afte attempting to open the event semaphore pointed to b *handle*. *Handle* is the value returned by a successful cal to SysCreateEventSem().

0	Event semaphore opened successfully
6	Error - invalid handle
8	Error - insufficient memory
87	Error - invalid parameter
123	Error - invalid name
187	Error - event semaphore not found
291	Error - too many opens

SysOpenMutexSem(handle) **(OBJ)**

Returns the DosOpenMutexSem return code afte attempting to open the mutex semaphore pointed to b *handle*. *Handle* is the value returned by a successful cal to SysCreateMutexSem().

0	Mutex semaphore opened successfully
6	Error - invalid handle
8	Error - insufficient memory
87	Error - invalid parameter
105	Error - semaphore owner died

123	Error - invalid name
187	Error - event semaphore not found
291	Error - too many opens

SysOpenObject(object_name, view, flag) **V3**

Returns 1 if the WPS object *object_name* was successfully opened on the Desktop; otherwise, returns 0.

Object_name can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

View specifies the view to be opened and can contain either a numeric value or the equivalent string. The function will pass all numeric values to the underlying wpOpen() or wpViewObject() function without testing the value for validity.

0	- DEFAULT
1	- ICON
2	- SETTINGS
3	- HELP
4	- RUNNING
5	- PROMPTDLG
121	- PALETTE

Flag can contain a 1 indicating that an existing view of an object can be opened on top of the Desktop (resurfaced) by calling the wpViewObject method or a 0 indicating that the view specified in *view* is to be opened using the wpOpen method. The following comment originated in the description of the wpOpen method:

"*In general, wpViewObject should be used instead of the wpOpen method. This is because wpViewObject takes into consideration the setting in the Object Open Behavior field on the Window page of the Settings notebook for the object. If a view of the object is already open, wpViewObject will depending on the setting of the Object Open Behavior field,* *either display the existing window for the object or create a new object."*

"In contrast, wpOpen always opens a new view of the object. Under certain circumstances this might be called for, but, under most circumstances, wpViewObject should be called instead."

SysOS2Ver()

Returns a string containing OS/2 version information in the form: x.xx.

```
Example:     2.10                    (pre V3)
             2.30                        (V3)
             2.40                        (V4)
```

SysPostEventSem(handle) **(OBJ)**

Returns the DosPostEventSem return code after posting the event semaphore indicated by *handle*. *Handle* is the value returned by a successful call tc SysCreateEventSem().

0	Event semaphore posted successfully.
6	Error - invalid handle
298	Error - too many posts
299	Error - already posted

SysProcessType() **(OBJ)**

Returns the type of process in which the REXX program is running. Functionally equivalent tc DOSSESSIONTYPE() (page 77).

0	Full screen protect mode session
1	Requires real mode (cannot occur)
2	VIO windowable protect mode session
3	Presentation Manager protect mode session
4	Detached protect mode process

SysPutEA(file, ea_name, ea_value)

Returns 0 if the *ea_name* extended attribute data i *ea_value* has been written to *file*; otherwise, returns th OS/2 return code of the failing function.

Note: *Standard system extended attributes ar terminated with a null character - '00'x.*

SysQueryClassList(stem)

Returns 0 after *stem* receives the entire set of registered classes and *stem*.0 is set to the number of registered classes. Each element of *stem* contains two words: *class name* and *module name* (i.e. DLL name). For some user defined classes, the full file system name is placed in *stem*.n.

```
Example:     WPObject  PMWP
             WPSystem  WPCONFIG
```

SysQueryEAList(file_name, stem) (OBJ)

Returns 0 after assigning the extended attribute names associated with *file_name* sequentially to *stem* with *stem.0* being assigned the number of extended attribute names retrieved; otherwise, returns the OS/2 error code of the failing function.

SysQueryProcessCodePage() (OBJ)

Returns the current code page for the process.

SysQueryRexxMacro(function_name) (OBJ)

Returns the position in the macrospace of *function_name* ('B' -before, 'A' - after) relative to external searching; otherwise, returns a null string (a zero-length string) if *function_name* is not located in the macrospace.

SysQuerySwitchList(stem) (OBJ)

Returns 0 after sequentially assigning the contents of the system switch list to *stem*. *Stem.0* will contain the number of entries in the switch list. A minimal switch list may contain entries such as:

```
Switch to
WarpCenter
Desktop
MUGLRQST.EXE
Workplace Shell
PMSPOOL.EXE
```

Note: The Warp 4 switch list contains a blank entry.

SysRegisterObjectClass(class_name, module_name)

Returns the resultant code from WinRegisterObjectClass: 1 if the class is registered successfully; otherwise, returns 0.

This function will register a new *class_name* and its associated *module_name (i.e. DLL name)* to the system.

SysReleaseMutexSem(handle) **(OBJ)**

Returns the return code from DosReleaseMutexSem after attempting to release ownership of the mutex semaphore identified by *handle*. *Handle* is the value returned by a successful call to SysCreateMutexSem().

0	Ownership released successfully
6	Error - invalid handle
288	Error - not owner

SysReorderRexxMacro(function_name, order) **(OBJ)**

Returns 0 after successfully reordering *function_name* in the REXX macrospace; otherwise, returns the return code from RexxReorderMacro. *Order* indicates the position in the macrospace of *function_name* ('B' -before, 'A' - after) relative to external searching.

0	Reorder successful
4	Error - function not found
8	Error - invalid position

SysRequestMutexSem(handle[, timeout) **(OBJ)**

Returns the return code from DosRequestMutexSem after successfully acquiring ownership of the mutex semaphore identified by *handle*. *Timeout* is a value, in microseconds, to wait on the semaphore. The default *timeout* is an indefinite wait. *Handle* is the value returned by a successful call to SysCreateMutexSem().

0	Ownership acquired successfully
6	Error - invalid handle
95	Error - interrupt
103	Error - too many requests
105	Error - semaphore owner died
640	Error - timeout

SysResetEventSem(handle) **(OBJ)**

Returns the return code from DosResetEventSem after attempting to reset the event semaphore identified by *handle*. *Handle* is the value returned by a successful call to SysCreateEventSem().

0	Event semaphore reset
6	Error - invalid handle
300	Error - semaphore already reset

SysRmDir(dirspec)

Returns one of the following codes resulting from the attempt to delete *dirspec*. Functionally equivalent to DOSRMDIR() (page 77).

0	Directory removed successfully.
2	Error - file not found.
3	Error - path not found.
5	Error - access denied.
16	Error - current directory.
26	Error - Not a DOS disk.
87	Error - invalid parameter.
108	Error - drive locked.
206	Error - filename exceeds range.

SysSaveObject(object_name, timing_flag) V3

Returns 1 if the WPS object *object_name* was successfully saved; otherwise, returns 0. File system objects (WPFileSystem) are saved in the file system's extended attributes and abstract objects are saved in the OS2.INI (user) file. Transient objects (WPTransient) cannot be saved.

Object_name can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

Timing_flag can be **0** (Boolean false - object is to be saved synchronously) or **1** (Boolean true - object is to saved asynchronously). If an asynchronous save is specified, the object will be saved on a separate thread ("lazy written"); this is the preferred method for saving. Otherwise, the object is saved on the user interface thread.

SysSaveRexxMacroSpace(file_name) (OBJ)

Returns the return code from RxxSaveMacroSpace after attempting to save all of the macrospace to *file_name*.

0	Save successful
2	Error - macros not found

3	Error - file requires an extension
5	Error - file error

SysSearchPath(env_path, file_name)

Returns the fully qualified name of *file_name* if it can be found by a search of the environment path identified by *env_path* (PATH, DPATH, etc.); otherwise, returns a null string. LIBPATH is not included in the search since it is not contained within the environment variables. Functionally equivalent to DOSPATHFIND() (page 75).

SysSetFileHandle(number) **(OBJ)**

Returns the DosSetMaxFH return code after attempting to set the number of available file handles in the current process to *number*. Functionally equivalent to DOSFILEHANDLES() (page 73).

0	Number of file handles set successfully.
8	Error - insufficient storage
87	Error - invalid parameter

SysSetIcon(file_name, icon_file_name)

Returns the resultant code from WinSetIcon: 1 if *icon_file_name* is successfully associated with *file_name*; otherwise, returns 0.

SysSetObjectData(object_name, setup_string)

Returns the resultant code from WinSetObjectData: 1 if *object_name* is successfully updated; otherwise, returns 0.

Object_name can be a WPS object ID (the unique string preceded with a '<' and terminated with a '>') assigned to the object when it was created (e.g. <WP_DESKTOP>) or a fully qualified file name. The predefined *object IDs* are shown in Section 4.1 beginning on page 107.

Setup_string must contain a WinCreateObject string which is comprised of a series of "key name=value" pairs that change the behavior of the object. "Key names" are separated by semicolons and "values" are separated by commas. Key names begin on page 126.

SysSetPriority(class, delta) (OBJ)

Returns the DosSetPriority return code after attempting to change the priority of the current session. The following priority classes can be specified in *class*:

0	No change, change priority *delta* only
1	Idle time priority class
2	Regular priority class
3	Time critical priority class
4	Foreground server priority class

Delta represents a change that can be made to the process priority level. It must contain an integer value in the range of -31 to +31. Functionally equivalent to DOSPRIORITY() (page 76).

0	Priority changed successfully
303	Error - invalid process ID
304	Error - invalid delta
305	Error - not descendant
307	Error - invalid class
308	Error - invalid scope
309	Error - invalid thread ID

SysSetProcessCodePage(code_page) (OBJ)

Returns the return code from DosSetProcessCp after altering the current code page for the process to *code_page*.

0	Code page set successfully
472	Error - invalid code page

SysShutDownSystem() (OBJ)

Returns 1 after initiating a successful OS/2 system shutdown; otherwise, returns 0.

SysSleep(seconds)

Returns 0 after the current session awakes from a sleep (wait) state for *seconds*. Functionally equivalent to DELAY() (page 66).

SysSwitchSession(session_name) **(OBJ)**

Returns 0 after attempting to switch the focus to *session_name*; otherwise, returns a non-zero value. *Session_name* must contain the session name as it appears in the window list and can be retrieved with the SysQuerySwitchList() function.

Note: *If* session_name *contains a non-switchable session, focus may be switched to the Window List.*

SysTempFileName(template[, {?̲ | filter}])

Returns a unique file or directory name which does not exist, according to template; otherwise, returns a null string. *Filter* is a single character whose occurrence in *template* is replaced with a numeric digit. *Template* may not exceed a length of 5. Functionally equivalent to DOSTEMPNAME() (page 78).

SysTextScreenRead(row, col[, length])

Returns a character string from the screen, beginning at *row* and *col*, of *length* or through the end of the screen, including CR & LF characters if the read spans across screen lines. The first column of the first row is referenced as 0 0.

SysTextScreenSize()

Returns two words indicating the size of the screen in rows and columns. Functionally equivalent to SCRSIZE() (page 94).

SysWaitEventSem(handle[, timeout]) **(OBJ)**

Returns the DosWaitEventSem return code after waiting on the event semaphore identified in *handle*. *Timeout* represents the time, in milliseconds, to wait on the semaphore. The default *timeout* is an indefinite wait.

0	Event has occurred
6	Error - invalid handle
8	Error - insufficient memory
95	Error - interrupted
640	Error - timeout

`SysWaitForShell( event[, query_flag] )` **(OBJ)**

Returns 1 (true) if *event* has occurred; otherwise, returns 0 (false). *Event* is a specific initialization event of the Workplace Shell and can be:

DESKTOPCREATED	Wait until Desktop has been created.
DESKTOPOPENED	Wait until Desktop has been opened.
DESKTOPPOPULATED	Wait until Desktop has been populated.

If *query_flag* is not specified, the function call will only return when the event has occurred. If *query_flag* contains the string **QUERY**, the function will return immediately and reflect the current status in the return value.

`SysWaitNamedPipe( name[, timeout] )`

Returns the resultant code from DosWaitNPipe after waiting for the named pipe *name* (\PIPE\pipename).

0	The named pipe is no longer busy.
2	The named pipe was not found.
231	The wait timed out before the pipe became available.

Timeout can be:

omitted	use default time.
-1	wait until pipe is no longer busy.
microsecs	wait for the specified time.

`SysWildCard( source_name, wild_card )` **(OBJ)**

Returns an edited file name resulting from the combination of *source_name* and *wild_card*. *Wild_card* is an editing pattern composed of file name wildcard characters ? and *. The edited result is the transformation resulting from the DosEditName function.

Example:

Source	Edit Pattern	Result
abc.src	*.bak	abc.bak
xyz.src	abc.*	abc.src
zoot.src	f???.bak	foot.bak

3. Other Rexx External Function Modules

REXXLIB and RXWINDOW are a combined, commercially available, external function package which provides an extensive array of enhancements in addition to those provided by REXXUTIL.

A fully functional demonstration version of REXXLIB that includes all the functions listed here can be downloaded from *http://www.quercus-sys.com*.

The information provided here is done so with the expressed permission of Quercus Systems. This handbook is not intended to replace the requirement of the Quercus documentation for the proper use of REXXLIB and RXWINDOW.

Some functions in REXXLIB duplicate those provided by REXXUTIL, but many are unique. REXXLIB, as part of Personal REXX, preceded OS/2 SAA REXX in the marketplace. Many of the OS/2 REXXLIB functions are identical to those provided with Personal REXX for DOS.

Quercus provides the *REXXLIB User's Guide* along with REXXLIB. In the event of any discrepancies between the material as presented here and the material as presented in the *REXXLIB User's Guide,* the Quercus publication should be assumed to be correct. Quercus also provided technical editing and guidance in the preparation of this entire handbook.

Quercus can be reached at:

P.O. Box 51218
Pacific Grove, California 93950
408-372-7399 (voice)
408-372-5776 (FAX)
http://www.quercus-sys.com
GO CIS:QUERCUS (CompuServe - 75300,2450)

REXXLIB contains the following functions by type:

- Mathematical functions including logarithms, exponentiation, trigonometric, hyperbolic and others.

- Stem and array functions including those which allow array sorting, array item insertion and deletion along with the ability to copy an entire array.

  The ability to manipulate compound arrays as well as functions which allow simple and compound arrays to be written to, and read from, disk. Also REXX arrays can be written to, as well as read from, ASCII disk files.

- OS/2 Information Services including: additional Extended Attribute support, enhanced file control functions as well as many miscellaneous functions such as the ability to determine the boot drive, place the OS/2 environment variables into a REXX array, determine the OS/2 file system type, and more.

- Hardware information and access functions which return data about the system that the program is running on, screen manipulation functions, and a special facility for reading from the keyboard as well as controlling the keyboard's typematic rate.

- An extensive collection of semaphore functions is included to provide an advanced Interprocess Communication capability.

Like REXXUTIL, REXXLIB and/or RXWINDOW must be loaded and REXX must be informed of their presence. This task is accomplished in a similar fashion to REXXUTIL and, once registered, each function is available to all other REXX sessions. REXXLIB can be registered with:

```
call RxFuncAdd 'RexxLibRegister', →
                     → 'REXXLIB', 'rexxlibregister'
call RexxLibRegister
```

It can be removed (unregistered) with:

```
call RexxLibDeRegister
```

RXWINDOW can be registered with:

```
call RxFuncAdd 'W_Register', 'RXWIN30', 'rxwindow'
call W_Register
```

It can be removed (unregistered) with:

```
call W_Deregister
```

3.1 REXXLIB Functions

ACOS(angle)

Returns the inverse of the cosine of *angle* (expressed in radians in the range of 0 to π). *Angle* must be a real number in the range -1 *angle* < 1 otherwise, 'NAN' (Not A Number) is returned. °

ARRAYCOPY(from_stem, to_stem[, from][, to][, ct])

Returns 1 if *from_stem* is successfully copied to *to_stem*; otherwise, returns 0. *From_stem* and *to_stem* are the source and target array stems. *From* and *to* represent the position in the respective arrays, with a default of 1, and *ct* indicates the number of array elements to be copied.

ARRAYDELETE(stem, from[, ct])

Returns 1 if one, or the number of elements indicated in *ct*, are successfully deleted from *stem* beginning at element *from*; otherwise, returns 0.

After successful deletion, the remaining initialized elements in the array are shifted downward.

ARRAYINSERT(from_stem, to_stem,[, from] →
→ [, to][, ct])

Returns 1 if one, or the number of elements indicated in *ct*, and located in *from_stem* are successfully inserted into *to_stem*. Source elements in *from_stem* begin at *from*, with a default of 1, and are inserted into *to_stem* at the position following *to*. *To* defaults to the number of elements in *to_stem* prior to the function being invoked.

ARRAYSEARCH(search_stem, result_stem, pattern →
→ [, options][, from][, count])

Returns the number of tails found while, at the same time, placing the indices of a REXX array (*search_stem*) into a new array (*result_stem*) for each index where the corresponding variable value matches a pattern.

Search_stem represents a REXX array to search. *Result_stem* is the stem name that identifies the output array. *Pattern* is a regular expression that is used to select tails.

Options indicates the type of search performed from the default, which is a case-insensitive search using a regular expression pattern. The value can be a string consisting of one or more of the following:

A	Select only items that do not satisfy the search conditions (Avoid).
C	Search should be case sensitive.
F	Exact match required between pattern and first substring of target, except possibly for case, no regular expressions.
S	Simple search that does not use regular expressions.
X	Exact match required between pattern and target, except possibly for case, no regular expressions

From represents the position in the first array at which the search begins. The default is 1. *Count* is the number of items to search. The default is all items, to the end of the array.

Notes: *Element 0 of the output array is set to the number of tails found.*

The input and output compound variables must be different.

ARRAYSEARCH is usually faster than CVSEARCH when the compound variable to be searched has only positive integral "subscripts", as is the case, for instance, with the result of FILEREAD.

You should avoid using a regular expression search by specifying 'S' in the options string if the pattern may contain any of the characters that have special meaning in regular expressions ("\", "^", "$", ".", ":", "", "+", "?", "[", and "]"), unless the characters are escaped by being preceded with "\". This is especially likely to be a problem if the pattern is entered as user input at run time or consists of a file name.*

ARRAYSORT(stem[, first][, n][, *sort controls*])
where *sort controls* can be up to 10 occurrences of
[start][, length][, order][, type]

Returns 1 if sort is successful; otherwise, returns 0. *Stem* indicates the name of an integrally-indexed REXX compound variable to be sorted. *First*, with a default of 1, represents the number of the element at which the sort is to begin and *n*, with a default of the number of elements from *first* up to the first uninitialized entry.

Start (default 1) indicates the position within the array element of the sort field.

Length (default 100) is the length of the sort field.

Order can be A for ascending or D for descending sequence.

Type can be C for case-sensitive character based fields for sort purposes, I for case-insensitive character based fields for sort purposes, or N for a numeric based fields for sort purposes.

ASIN(angle)
Returns the inverse of the sine of *angle* (expressed in radians in the range of -π/2 to π/2). *Angle* must be a real number in the range -1 ≤ *angle* < 1 otherwise, 'NAN' (Not A Number) is returned. °

ATAN(angle)
Returns the inverse of the tangent of *angle* (expressed in radians in the range of -π/2 to π/2). *Angle* must be a real number in the range -1 ≤ *angle* < 1 otherwise, 'NAN' (Not A Number) is returned. °

ATAN2(x, y)
Returns the inverse of the tangent of *x/y* (expressed in radians in the range of -π/2 to π/2). If both *x* and *y* are 0, 'NAN' (Not A Number) is returned. °

CHARSIZE()
Returns the height of the character mode text box. °

COS(angle)

Returns the cosine of *angle* (expressed in radians). °

COSH(angle)

Returns the hyperbolic cosine of *angle* (expressed in radians). °

CURSOR([row][, col])

Returns the current cursor position on the screen as two whole numbers, row and column°, and optionally moves the cursor to the position specified in *row* and *col*. The first column of the first row is returned as 1 1. Similar in function to SysCurPos (page 42).

Example: 4 1

CURSORTYPE([start], [end], [visibility])

Returns the current cursor type before replacing the cursor with one which begins at *start* and ends at *end* pixel lines within the character box.

Visibility can be 1 to make the cursor visible or 0 to make it invisible. If omitted, cursor visibility does not change.

CVCOPY(from_stem, to_stem)

Returns 1 if the elements of the compound variable *from_stem* are successfully copied to *to_stem*; otherwise, returns 0.

CVREAD(file_id, to_stem)

Returns 1 following the successful read of *file_id*, a file containing the elements of a compound variable and created with CVWRITE. The elements are read into *to_stem*. *File_id* must be a fully qualified file name.

To_stem is initialized prior to the read occurring.

CVSEARCH(stem, tail_stem, pattern[, options])

Returns the number of tail elements (those following the first '.') in *stem* which are placed into *tail_stem* as a result of searching *stem* with *pattern*. *Options* can be 'C' (non-default) to indicate a case sensitive search. °

See the *REXXLIB User's Guide* for a description of *pattern*.

CVTAILS(stem, tail_stem[, pattern][, options])

Returns the number of tail elements (those following the first '.') in *stem* which are placed into *tail_stem*. If *pattern* is used to select tails, only the tail portion of the compound variable is used for matching. *Options* can be 'C' (non-default) to indicate a case sensitive search. °

See the *REXXLIB User's Guide* for a description o *pattern*.

CVWRITE(file_id, from_stem)

Returns 1 indicating the *from_stem* has been successfull written to *file_id*; otherwise, returns 0. *File_id* must be a fully qualified file name.

DATECONV(date[, input_format[, output_format])

Returns a converted date, which is in *input_forma* format (B, C, D, E, J, N, O, S, or U), to *output_forma* (B, C, D, E, J, M, O, S, U or W). The formats are the same as those used for the DATE() built-in functior (page 21 for a description of these formats).

DELAY(duration)

Returns 1 after suspending execution for *duratior* seconds, which can be a fraction. Suspension o execution is accurate to about one tenth of a second Functionally equivalent to SysSleep() (page 57).

DETAB(string[, space_count])

Returns *string* with all tab characters ('09'x) removed anc replaced with an appropriate number of spaces *Space_count*, with a default value of 8, represents the number of spaces between tab stops.

Notes: *Tab characters are often inserted in ASCII datc files by text editors and are also common in file: transferred from Unix systems. Unlike Unix, OS/2 does not ordinarily equate tabs to spaces, so i may be necessary to remove them with DETAB*

DOSAPPTYPE(file_id)

Returns 3 words, each containing a whole number indicating the type of executable file *file_id* is. Return: 0 0 0 for a non-executable file.

Word 1:

0 -	application type not specified in file.
1 -	program is OS/2 full-screen only.
2 -	program can execute in PM text-mode window.
3 -	application is a PM program.
16 -	program is a DLL.
32 -	program is DOS only.

Word 2:

1 - indicating that the program has been "bound"; otherwise, returns 0.

Word 3:

1 - indicates the application is a 32-bit program; otherwise, returns 0.

DOSBOOTDRIVE()

Returns the one character disk letter, in uppercase, of the system boot drive.

DOSCD([drive_letter])

Returns the name of the current directory on *drive_letter* or on the current drive. Returns a null string if *drive_letter* is not available.

DOSCHDIR(path_name)

Returns 1 if the current directory on a particular drive is successfully changed to *path_name*; returns 0 otherwise.

DOSCHMOD(file_name[, turn_on[, turn_off])

Returns 1 indicating that attributes of the fully qualified *file_name* have been successfully changed; returns 0 otherwise.

Turn_on and *turn_off* represent strings containing the letters Hidden, System, Read-only or Archive which are the only attributes which can be changed with DOSCHMOD.

DOSCOMMANDFIND(command_file)

Returns the fully qualified name of *command_file* if it is located in the current directory or system path; otherwise, returns a null string.

If either a drive or directory is included in *command_file*, then only that drive or directory is searched. A file is considered a command file if it has an extension of .COM, .EXE or .CMD and a match is found in the same priority.

DOSCOPY(source_name [,target_name] →
→ [, mode][, options])

Returns 0 if *source_name* is successfully copied to *target_name*; otherwise, returns an error code. Possible error codes include:

2	source file not found
3	source or target path not found
5	target file exists but 'A' or 'R' mode not specified
32	sharing violation for source or target file
108	source or target drive locked
112	disk full
206	invalid source or target file name
267	source name is a directory
282	extended attributes not supported for target

Mode is a letter ('R', 'A', or 'N') which indicates whether the target file, if it already exists, should be Replaced, Appended, or left unchanged.

Options contains a string consisting of one or more of the following:

'V' Verify the data after copying and indicate an error if the copied data does not match.

If the target of the copy operation is not specified, it is a file in the current directory having the same name as the source of the copy.

Wildcard characters ('*' and '?') are not allowed in the source or target names.

By default, a copy fails if the target already exists. This is a mode of 'N'. If the mode is 'A', and the target exists, the data from the source is appended to the target. If the mode is 'R' and the target exists, it is replaced. If the target is read-only, the copy fails regardless of mode. Attributes of the source (last modification date, etc.)

are copied to the target unless the target already exists and is being appended.

Extended attributes are copied from the source to the target unless the target already exists and is being appended. If the volume that holds the target does not support extended attributes (e. g. a floppy), the data will be copied without extended attributes.

It may be faster to use the DOSCOPY function instead of the system COPY command if the default execution environment is not CMD.EXE, since the need to start a new process can be avoided. It is also safer, since unlike the COPY command, DOSCOPY does not replace an existing target unless explicitly requested to do so.

If an error occurs during copying, the target file is deleted, unless it was being appended, in which case it is restored to its original size.

DOSCREAT(file_name)

Returns 1 if *file_name* is successfully created. When file_name is created or, if *file_name* already exists, its length is set to zero.

DOSDEL(file_name)

Returns 1 if *file_name* is successfully deleted; returns 0 otherwise. Functionally equivalent to SysFileDelete() (page 44).

DOSDIR([file_id][, output][, search][, mask] →
→ [, position]

Returns the directory information described below for *file_id* or a null string if *file_id* is not located.

If *file_id* contains drive or path information, the specified drive or directory is searched; otherwise, the current drive of the current directory is searched. If *file_id* contains wildcards, the first matching file is located. If *file_id* is omitted, it is assumed that a previous call was made to DOSDIR specifying a *file_id* with wildcard characters, and the next matching file is found.

The returned string contains:

- The name and extension of *file_id* in the format *filename.ext*.
- The size of *file_id* in bytes. °
- The date of the last update of *file_id* in the form *mm/dd/yy*.
- The time of the last update of *file_id* in the form *hh:mm:ss*.
- An ordered group of characters indicating the file attributes associated with *file_id* The presence of the respective character indicates the attribute is set; otherwise, the position contains a dash.

R	read-only
H	hidden file
S	system file
D	directory entry
A	archive bit is set
-	none of the above attributes is set

```
Example:     OS2.INI 115520 09/29/93 12:28:42 RA
             EDIT.CMD  2526 01/11/93 12:29:00 -
```

If *output* is omitted or is null, all of the above information is returned by DOSDIR; otherwise, *output* can contain any of the following letters, in any order, which result in just the specified fields to be returned and in the same order as the respective letters specified in *output*.

N	*file_id*'s name and extension are returned
S	*file_id*'s size is returned
D	*file_id*'s date is returned
T	*file_id*'s time is returned
A	*file_id*'s attribute string is returned

If *source* is omitted or null, DOSDIR searches only for "normal' files. Hidden files, system files and directory files are ignored. *Search* can contain an unordered string of the following characters that cause the respective additional entries to be included in the search:

H	hidden files are included
S	system files are included
D	directory files are included.

If *file_id* is not specified and DOSDIR is searching for the next match of a previously-specified pattern, *search* is ignored.

R	restricts the search to read-only files
H	restricts the search to hidden files
S	restricts the search to system files.
D	restricts the search to directory files
A	restricts the search to files with the archive bit set

With *file_id* omitted, DOSDIR normally searches for the next file which matches the last used pattern. The *position* operand can be used to cause a search for the next file matching a previously used pattern, allowing intervening calls to DOSDIR using other patters. The DOSDIRPOS function is used to save the current position in a directory search.

`DOSDIRCLOSE( position )`

Returns 0 if DOSDIR is successfully notified that a file search is complete; otherwise, returns 1.

`DOSDIRPOS()`

Returns a string representing the current DOSDIR directory position status. A later call to DOSDIR can specify the status via its *position* argument to resume processing a directory search at the current position.

`DOSDISK( option[, drive_letter )`

Returns information about *drive_letter* or the current drive if *drive_letter* is omitted; otherwise, returns -1 if *drive_letter* is invalid. The information returned is dependent on *option*.

A	The number of available clusters on the disk.
B	The number of bytes per disk sector.
C	The number of clusters on the disk.
F	The number of free (unused) bytes on the disk.
S	The number of sectors per disk cluster.
T	The total number of bytes on the disk.
U	The total number of bytes in use on the disk.

DOSDRIVE(new_drive_letter)

Returns the letter of the current drive before optionally establishing *new_drive_letter* as the current default drive.

DOSEALIST(file_id[, name_stem[, value_stem] →
→ [, flag_stem])

Returns the number of extended attributes that exist for *file-name*. -1 is returned if *file_id* does not exist or if an error occurs retrieving the extended attributes. °

Name_stem is the name of a compound variable which receives the name of the extended attribute. *Value_stem* is the name of a compound variable which receives the value of the extended attribute. *Flag_stem* is the name of a compound variable which receives the flag byte of the extended attribute.

DOSEASIZE(file_id)

Returns the total size of the extended attributes belonging to *file_id* or -1 if *file_id* does not exist or an error occurs retrieving the extended attributes. °

DOSEDITNAME(source, pattern)

Returns a file name character string based on *source* and *pattern*. Characters are copied, one at a time, from *pattern* to the result string. Each character in *pattern* copied causes a pointer to advance in *source*. The operation stops when *pattern* is exhausted.

If '?' is found in *pattern*, the current character in *source* is copied unless a period or the end of *source* has been encountered.

If '*' is found in *pattern*, characters in *source* are copied until the character following the '*' in *source* is reached or *source* is exhausted.

If '.' is found in *pattern*, the *source* pointer is advanced to the character following the '.'.

DOSENV(environment_variable_name)

Returns the value associated with *environment_variable_name* or a null string if *environment_variable_name* is not found. Functionally equivalent to the VALUE() function with OS2ENVIRONMENT specified as the selector (page 32).

DOSENVLIST(list_stem)

Returns the number of OS/2 environment variables. Each environment variable is copied to *list_stem*.

DOSENVSIZE()

Returns two whole numbers indicating the total size of the current environment area and the number of free bytes in the current environment area.

DOSFDATE(file_id[, new_date][, new_time])

Returns 1 if *new_date* and/or *new_time* have successfully replaced the date/time indicating the last update of *file_id*; otherwise, returns 0. *New_date* is specified in the "sorted" date format (DATE('S') - see page 21) and must be specified as 8 digits, without punctuation, and represents *yyyymmdd*. *New_time* must be 6 digits and represents *hhmmss* (TIME('N') without colons).

DOSFILEHANDLES([handles])

Returns the maximum number of system file handles that may be opened in a given process and optionally changes the number to *handles*. Similar in function to SysAddFileHandle (page 37).

Note: *File handles are used every time a file is accessed with REXX I/O functions such as CHARIN / CHAROUT, LINEIN / LINEOUT, CHARS, LINES, and STREAM. Other services invoked indirectly from REXX (e.g. SysGetMessage() or other API routines) may also create open file handles. At least 3 handles are usually in use for the standard input stream, standard output. stream, and standard error stream.*

DOSFILEINFO(file_id, option)

Returns the following depending on *option*:

A	Returns the date and time the file was last accessed
C	Returns the date and time the file was created.
S	Returns the allocated size of the file. °

Date and time are returned as two words in the form mm/dd/yy hh:mm:ss. *Options* **A** & **C** are only valid fo HPFS disks. If *file_id* specifies a file on a FAT disk anc *option* is either **A** or **C**, the string returned is: 00/00/8(00:00:00.

DOSFILESYS([drive_letter],)

Returns 'FAT', 'HPFS', 'CDFS' or 'LAN' indicating the file system on *drive_letter* or on the current drive i *drive_letter* is omitted. LAN is returned for all pee drives on Warp Connect. Functionally equivalent tc SysFileSystemType() (page 45).

DOSFNAME(file_name)

Returns the fully qualified name of *file_name* including drive, path, name and extension for *file_name* on the current or specified directory or a null string if *file_name* is invalid.

DOSFSIZE(file_id, [new_size])

Returns the actual size of the fully qualified *file_id* optionally changing the size of *file_id* to *new-size*. Returns -1 if *file_id* does not exist. °

DOSISDEV(file_name)

Returns 1 if *file_name* is a device name such as: CON, PRN, LPT1, NUL, etc.; otherwise, returns 0.

DOSISDIR(file_id)

Returns 1 if *file_id* represents an existing directory; otherwise, returns 0.

Note: *File_id should not contain a trailing slash.*

DOSISFILE(file_id)

Returns 1 if *file_id* represents an existing file. Returns 0 if *file_id* does not exist, is a device or is a named pipe.

`DOSISPIPE( file_name )`

Returns 1 if *file_name* represents an existing named pipe; otherwise, returns 0.

`DOSKILLPROCESS( pid[, option] )`

Returns 1 if the system successfully flagged the process identified in *pid* for termination; otherwise, returns 0. An *option* of 'P' indicates that there is no restriction on the process to be terminated. An *option* of 'T' indicates the *pid* must be the current process or one of its descendants.

`DOSMAKEDIR( path_name )`

Returns 1 if the directory specified in *path_name* is successfully created or currently exists; otherwise, returns 0. This function differs from SysMkDir and DOSMKDIR in that it will create any necessary intermediate directories.

`DOSMAXPATH()`

Returns the maximum acceptable length of a fully qualified path name.

`DOSMKDIR( path_name )`

Returns 1 if the fully qualified *path_name* results in the successful creation of the named directory. Functionally equivalent to SysMkDir() (page 49).

`DOSPATHFIND( file_name[, environment] )`

Returns the fully qualified name of *file_name*; otherwise, returns a null string. *Environment*, with a default of PATH, indicates an environment variable which contains a list of directories. DOSPATHFIND will not use LIBPATH since it is not an environment variable. Functionally equivalent to SysSearchPath() (page 56).

`DOSPID()`

Returns the process id (PID) of the current process. This is a number which uniquely identifies the process within the system.

DOSPIDLIST(pid_stem[, name_stem[, parent_stem →
→ [, session_stem])

Returns the number of processes found and placed into the compound variable *pid_stem*. °

Name_stem is a compound variable that receives the names of these processes. *Parent_stem* is the name of the compound variable which receives the list of parent process identifiers. *Session_stem* is the name of a compound variable that receives the list of session identifiers.

DOSPRIORITY([delta], [class], pid)

Returns two whole numbers indicating the priority level and the priority class of thread 1 of the process specified in *pid*. If *pid* indicates a descendant of the requesting task, the *delta* and *class* can be specified to alter the current values. Functionally equivalent to SysSetPriority() (page 57).

Delta is a signed number from -31 to +31 that indicates the requested change to priority. 0 indicates no priority change is to occur.

Class is a number from 1 to 4 indicating a new priority class.

DOSPROCINFO(option[, pid])

Returns information about *pid* depending upon *option.* If *pid* is omitted, information is returned for the current process.

N	Returns the fully qualified name of the program associated with *pid*.
P	Returns the process identifier of the parent process.
S	Returns the session identifier of the process.

DOSRENAME(file_name_1, file_name_2)

Returns 1 if *file_name_1*, which must contain a fully qualified file or directory name, is successfully renamed to the new name in *file_name_2*; otherwise, returns 0.

DOSRMDIR(directory_name)

Returns 1 if *directory_name* is successfully removed; otherwise, returns 0. Functionally equivalent to SysRmDir() (page 55).

DOSSESSIONTYPE()

Returns a number indicating the type of session the function was invoked in. Functionally equivalent to SysProcessType() (page 52).

0 -	full-screen.
2 -	VIO window
3 -	PM session
4 -	detached session

```
DOSSWITCHLIST( title_stem[, pid_stem]  →
               → [, session_stem][, handle_stem]  →
               → [, type_stem][, visibility_stem] )
```

Returns a whole number indicating the number of switch list entries found and places information about the sessions in the system "switch list" (Window list) into one or more compound variables.

Title_stem:	A stem name for the compound variable that receives the title of the session.
Pid_stem:	A stem name for the compound variable that receives the list of process identifiers.
Session_stem:	A stem name for the compound variable that receives the list of session identifiers.
Handle_stem:	A stem name for the compound variable that receives the list of window handles.
Type_stem:	A stem name for the compound variable that receives the list of program types.
Visibility_stem:	A stem name for the compound variable that receives the list of visibility flags.

Not all processes listed by DOSPIDLIST() will be returned by DOSSWITCHLIST(). Only those processes which are included in the switch list are returned.

Conversely, not all processes listed by DOSSWITCHLIST() will be returned by DOSPIDLIST(). For instance, processes corresponding to DOS or Windows sessions are not shown by DOSPIDLIST().

The session title is the string that occurs in the title bar of the session's window and in the switch list. All "white space" characters between words in the title are removed, except for a single blank.

Element 0 of each compound variable array is set to the number of elements in the array.

The program type is a number which indicates the type of the session. It may be:

0 -	OS/2 full screen
2 -	OS/2 window
3 -	PM application
4 -	DOS or Windows full screen
7 -	DOS or Windows window

The visibility flag is a number that indicates whether the item is actually visible in the switch list. It may be:

1 -	invisible
2 -	grayed
4 -	fully visible

DOSTEMPNAME(name_pattern)

Returns a valid file name, unique to the current or default drive, derived from *pattern*; otherwise, returns a null string if a unique name can not be created. Functionally equivalent to SysTempFileName() (page 58).

DOSTID( )

Returns the thread id (TID) of the current thread. This is a number which uniquely identifies the thread within the current process.

DOSVOLUME([drive_letter])

Returns the volume label of *drive_letter* or the current drive. If a disk is unlabeled, a null string is returned.

DOSVOLINFO([disk], [option])

Returns information about a specified disk volume. *Disk* is the letter that identifies the disk for which information is requested. Default is the current disk. *Option* is a letter that indicates the type of information, which can be:

F	File system type (e. g. FAT, HPFS, CDFS, LAN).
L	The volume label.
S	The volume serial number (default).

Note: The volume serial number is a unique identifier created when the volume is formatted. It is returned in the form xxxx-xxxx, where x is a hex digit.

The volume label is the user-specified identifier associated with the volume by the FORMAT or LABEL command. This is the same information returned by DOSVOLUME.

The file system type is the same information returned by DOSFILESYS.

ENTAB(string, [space_count])

Returns *string* with *space_count* consecutive blanks in *string* replaced with tab characters ('09'x). *Space_count* defaults to 8.

Note: Replacing blanks with tabs can make some strings considerably shorter. This may be significant when creating large ASCII TEXT files.

ERF(x)

Returns the error function ($2\pi^{-1/2} \int_0^X e^{-t**2}\, dt$). °

ERFC(x)

Returns the complement of the error function (1 - ERF(x)). °

EVENTSEM_CLOSE(name)

Returns 0 if event semaphore *name* is successfully closed; otherwise, a return code identifying the type of error.

Note: *This function should be used when you are done with an event semaphore to release associated storage resources.*

EVENTSEM_CREATE(name[, options])

Returns 0 if event semaphore *name* was successfully created; otherwise, a return code identifying the type of error. An *option* of 'P' (non-default) causes the semaphore to be created in a "posted" state.

Note: *This function should be invoked once and only once to define the event semaphore to the system.*

When an event semaphore is in the posted state, the event that it represents is considered to have occurred and EVENTSEM_WAIT will not wait.

EVENTSEM_POST(name)

Returns 0 if event semaphore *name* is successfully posted; otherwise, a return code identifying the type of error.

Note: *When an event semaphore is in the posted state, the event that it represents is considered to have occurred and EVENTSEM_WAIT will not wait.*

EVENTSEM_QUERY(name)

Returns the number of times event semaphore *name* has been posted since it was last reset. If an error occurs, a negative number is returned that identifies the type of error.

EVENTSEM_RESET(name)

Returns the number of times event semaphore *name* has been posted since it was last reset if the operation was successful; otherwise, a negative number is returned that identifies the type of error.

Note: *When an event semaphore is reset, the event it represents is considered not to have occurred and EVENTSEM_WAIT will wait.*

EVENTSEM_WAIT(name[, timeout])

Returns 0 after event semaphore *name* is posted; otherwise, returns a code identifying the type of error. Return does not occur until *timeout* milliseconds have elapsed. The default for *timeout* is -1 indicating an indefinite wait.

EXP(x)

Returns the exponential function of x - the value of e^x (e=2.718281828...). °

FILECRC(file_name)

Returns the 32-bit CRC (cyclic redundancy check) for *file_name* as 8 characters.

FILEREAD(file_name, stem[, count][, option] →
→ [, line_end[, start_pos])

Returns the number of lines read from *file_name* and placed into the compound variable *stem*. A negative return code indicates an error reading the file (e. g. invalid name, file not found).

The presence of *count* results in *count* lines being read rather than the entire file. The *option*, **E**, will result in the read stopping if an end-of-file character ('1A'x) is encountered. °

Line_end defines the string used to determine the end of a line and can be one of the following:

<u>**LFONLY**</u>	Line feeds and carriage return - line feed pairs.
CRONLY	Only carriage returns and carriage return - line feed pairs.

CRORLF	Line feeds, carriage returns, or carriage return - line feed pairs.
CRANDLF	Only carriage return - line feed pairs.

Start_pos indicates the relative position in the file at which to begin reading. (The first byte of the file is 1.)

Note 01: Element 0 of the compound variable array (stem) *is set to the number of lines read.*

Note 02: There is no particular limit on the length of any line which may be read.

```
FILESEARCH( file_name, pattern, stem[, options] →
                        → [, zone1][, zone2][, count] →
                             [, line_end][, start_pos] )
```

Returns the number of lines from *file_name* which, as a result of matching *pattern* subject to *options*, are placed into *stem*. A negative return code indicates an error reading the file (e. g. invalid name, file not found). °

Pattern is the regular expression pattern that describes the search string. *Options* can be one or more of:

A - Avoid; select only lines which do not match *pattern*.
C - Case sensitive matching (non-default).
N - Number. Precede each line with its relative line number.
E - EOF sensitive. The search will stop if an end-of-file character ('1A'x) is encountered.

Zone1 and *zone2*, with a default of 1 and end-of-line respectively, define the scope of the search argument in each line. *Count* indicates the number of lines to search. The default is the whole file.

Line_end defines the string used to determine the end of a line and can be one of the following:

LFONLY	Line feeds and carriage return - line feed pairs.
CRONLY	Only carriage returns and carriage return - line feed pairs.
CRORLF	Line feeds, carriage returns, or carriage return - line feed pairs.

CRANDLF Only carriage return - line feed pairs.

Start_pos indicates the relative position in the file at which to begin reading. (The first byte of the file is 1.)

*Note 01: Element 0 of the compound variable array (*stem*) is set to the number of lines read.*

FILEWRITE(file_name, stem[, option][, count] → → [, start])

Returns 1 if the contents of the compound variable *stem* is successfully written to *file_name* as an ASCII file. *Count* and *start* have the respective defaults of the number of elements in the compound variable and 1. *Option* can be one of the following:

A - Append the file, if it exists.
<u>R</u> - Replace the file, if it exists.
N - Nothing is to occur if the file exists.

Note: The file is closed after each call to the function.

GAMMA(x)

Returns the gamma function for $x>0$ or a null string for $x\leq0$. °

GREP(pattern, string[, options])

Returns two numbers, separated by a blank, which represent the position in *string* and the length of the substring matched by *pattern* or 0 0 if there is no match. Pattern is a regular expression pattern and *option* can be 'C' indicating a case sensitive (non-default) search. °

INKEY([wait_option[, keyboard_option])

Returns an encoded string that represents which key was pressed. A list of all of the possible combinations of keys and the data returned by this function for each key is listed in Appendix B (page 211). Functionally equivalent to SysGetKey() (page 47).

Wait_option is either <u>W</u>ait or No wait for a key to be pressed. If no wait is specified and a key has not been pressed, a null string is returned.

Keyboard_option, with a default of F indicates that keys with dual representation (i.e. the leading byte of the 2 byte scan code is 'E0'X) are folded into the same representation as their other scan code (i.e. the leading byte is changed to '00'X). The *keyboard_option* **E** will result in the exact scan code being returned.

LOG(x)

Returns the natural logarithm of *x* for *x>0*; 'INFINITY' for *x=0*; or 'NAN' (Not A Number) for *x<0*. °

LOG10(x)

Returns the logarithm (base 10) of *x* for *x>0*; 'INFINITY' for *x=0*; or 'NAN' (Not A Number) for *x<0*. °

LOWER(string)

Returns *string* in lowercase.

MACROADD(function, file_name[, position])

Returns 1 if *function* is successfully added to the macrospace; otherwise, returns 0. *File_name* is the fully qualified name of the REXX program file. *Position* is either **B**, indicating that the macrospace should be searched before the list of registered functions and functions on disk; or, **A** indicating that the macrospace should be searched after a search is made of registered functions and functions on disk.

MACROCLEAR()

Returns 1 indicating that the macrospace has been cleared of all REXX procedures; otherwise, returns 0.

MACRODROP(function)

Returns 1 if *function* is successfully removed from the macrospace; otherwise, returns 0.

MACROLOAD(file_name[, name_stem])

Returns 1 if some or all of the named functions in the compound variable, *name_stem*, and located on the saved macrospace file, *file_name*, are successfully loaded to the macrospace.

If *name_stem* is omitted, all functions in *file_name* are loaded. If *name_stem* is present, *name_stem*.0 must contain the number of procedure names in the array.

MACROQUERY(function)

Returns **B** if the *function* search position in the macrospace is before external functions, **A** if the *function* search position is after external functions, or null if *function* is not found in the macrospace.

MACROREORDER(function, position)

Returns 1 if search order of *function* in the macrospace is successfully changed; otherwise, returns 0. Position is either **B**, search the macrospace before searching external function packages or disk files; or **A**, search the macrospace afterward.

MACROSAVE(file_name[, name_stem])

Returns 1 if some or all of the named functions in the compound variable, *name_stem*, are successfully saved on the macrospace file *file_name*. If *file_name* exists, it is replaced.

If *name_stem* is omitted, all functions in the macrospace are saved to *file_name*. If *name_stem* is present, *name_stem*.0 must contain the number of procedure names in the array.

MUTEXSEM_CLOSE(name)

Returns 0 if mutual exclusion semaphore *name* is successfully closed; otherwise, a return code identifying the type of error.

MUTEXSEM_CREATE(name[, options])

Returns 0 if mutual exclusion semaphore *name* was successfully created; otherwise, a return code identifying the type of error. An *option* of 'O' (non-default) causes the semaphore to be created in an "owned" state.

MUTEXSEM_QUERY(name)

Returns a string of three numbers indicating: the number of times the mutual exclusion semaphore, *name*, has been requested by its owner less the number of times it has been released; the process id of the owner; the thread id of the semaphore owner.

If the semaphore is not currently owned, all three numbers returned will be 0 and if an error occurs, a negative number is returned identifying the type of error.

MUTEXSEM_RELEASE(name)

Returns 0 if the ownership of a mutual exclusion semaphore, *name*, is released; otherwise, a number identifying the type of error.

MUTEXSEM_REQUEST(name[, timeout])

Returns 0 if the ownership request for a mutually exclusive semaphore, *name*, is successful; otherwise, a return code identifying the type of error. Return does not occur until *timeout* milliseconds have elapsed. The default for *timeout* is -1 indicating an indefinite wait.

NMPIPE_CALL(name, message[, max][, timeout])

Returns a string sent by the server, across pipe *name*, in response to *message*. *Max* (default 4096) is the maximum length of the expected reply and *timeout* is the length of time, in milliseconds, to wait for the pipe to become available.

NMPIPE_CLOSE(name)

Returns 0 if *name* is successfully closed; otherwise, an error code.

NMPIPE_CONNECT(name)

Returns 0 if a connection with *name* is successfully established; otherwise, an error code.

NMPIPE_CREATE(name[, type][, mode][, wait] →
→ [, out][, in])

Returns the pipe handle, in the form of a number, if *name* was successfully created; otherwise, a negative number identifying the error.

Type:	'<u>**B**</u>' for a byte stream pipe; '**M**' for a message pipe.
Mode:	Read mode if the pipe is message pipe: '<u>**B**</u>' if the read mode is byte; '**M**' if the read mode is message.
Wait:	'<u>**W**</u>' if reads should block waiting for data; '**N**' if reads should not block.

Out and *in* indicate the size, in bytes, of the output and input buffers, respectively (default 4096).

NMPIPE_DISCONNECT(name)

Returns 0 if the server was able to terminate a communication session with *name*; otherwise, returns an error code.

NMPIPE_OPEN(name[, mode][, wait][, in])

Returns 0 if a communication session with *name* was opened successfully; otherwise, returns an error code.

Mode:	Read mode if the pipe is a message pipe: '<u>B</u>' if the read mode is byte; 'M' if the read mode is message.
Wait:	'<u>W</u>' if reads should block waiting for data; 'N' if reads should not block.

In indicates the size, in bytes, of the input buffer (default 4096).

NMPIPE_READ(name[, count])

Returns the data string from the pipe *name*. The number of bytes to be returned can be specified in *count* or will default to the pipe's input buffer size.

NMPIPE_TRANSACT(name, message[, max])

Returns a data string sent by the server in response to *message*. The *max* size of the reply expected, with a default of 4096.

NMPIPE_WRITE(name, data)

Returns the number of characters of *data* not written to *name*. A negative return code indicates an error.

PARSEFN(file_name)

Returns a string consisting of 4 uppercase words resulting from parsing *file_name*:

Drive letter (no colon)
Path specification
file name
file extension

Components not present are replaced with a dash "-". *File_name* is not checked for validity, just for syntactical correctness. Similar in function to FILESPEC (page 22).

`Example:     C  \ABC\  XYZ  EXT`

PCCOPROCESSOR()

Returns 1 if a hardware math coprocessor exists; otherwise, returns 0.

PCDISK(option[, drive_letter])

Returns information, depending upon *option*, about a fixed or floppy disk drives indicated in *drive_letter*, or the current drive.

N	Returns the number of fixed and floppy drives.
H	Returns the number of heads (read-write surfaces) (i.e. the number of tracks per cylinder).
C	Returns the number of disk cylinders.
S	Returns the number of sectors per track.

A drive letter can only be specified for the **H**, **C**, and **S** functions and these *options* only apply to hard disks.

PCFLOPPY()

Returns the number of floppy drives installed.

PCMODEL()

Returns the system (BIOS) defined model number.

`Example:     252`

PCPARALLEL()

Returns the number of parallel ports installed.

PCRAM()

Returns the number of 1K (1,024 bytes) of RAM (random access memory - real storage) installed.

PCSERIAL()

Returns the number of serial ports installed.

PCSUBMODEL()

Returns the system (BIOS) defined submodel number.

`Example:     1`

PCVIDEOMODE()

Returns 4 numeric words which describe information about the current screen mode. Word 1 is 1 if the function is successful otherwise word 1 is -1.

Word 2 is the number of bits of color information which can be displayed for each pixel.

Words 3 and 4 contain the horizontal and vertical resolution of the current screen in pixels.

```
Example:     1  4  720  400
```

POW(x, y)

Returns the value of *x* raised to the *y* power (x^y) if *x* is non-negative and *y* is an integer; otherwise, returns 'NAN' (Not A Number). °

PMPRINTF(string)

Returns 1 after writing *string* to the PMPRINTF window.

Note: *You must have the PMPRINTF program to use this function. The PMPRINTF program is started in a separate window just like any other OS/2 application. The PMPRINTF program is IBM EWS (Employee Written Software) that is available from many OS/2 Bulletin Boards and information services. It can also be found on the* OS/2 Warp Version 3 Unleashed *companion CD-ROM (ISBN 0-672-30545-3).*

REXXLIBDEREGISTER()

Returns 1 and removes the definitions of all REXXLIB functions from the operating system.

REXXLIBREGISTER()

Returns 1 and defines all REXXLIB functions to the operating system.

REXXLIBVER()

Returns the current REXXLIB version number.

```
REXXRUN( type, source[, result]  →
                          → [, arg-1]...[, arg-n] )
```

Returns the return code from the REXX interpreter resulting from running an external REXX program from a file, the REXX macrospace, a source code string, or a tokenized source code string. This will be 0 if the program ran to completion. A positive number indicates a problem starting the interpreter. A negative number is a program execution error (the negative of the REXX error code - see Appendix D on page 215).

Type is the type of source code, which can be:

F - Source code is in a file.
S - Source code is in a string.
T - Source code is in a tokenized string.
M - Source code is in the macrospace.

Source depends on the value of *type*. For each type below, *source* can be:

F - the actual program code
S - source code in a string
T - tokenized source code
M - the name of a function in the macrospace

Result is the name of a REXX variable to receive the value returned, if any, by the program.

Arg-i are the arguments to be passed to the program.

Source code may be supplied in a REXX string (type 'S'). This should be in a form just as if the program were read from a file, with lines separated by carriage return and line feeds, and statements on the same line separated by semicolons.

Tokenized code is created as the output of the TOKENIZEFILE or TOKENIZESTRING functions.

Programs are loaded into the macrospace with the MACROADD or MACROLOAD functions.

A variable name may be specified to receive the value returned by the external program. If no value i returned, the variable will be dropped. Avoid using a variable named RESULT, since this variable is set by REXX itself after a CALL statement.

There are several other ways to invoke external REXX programs, but all have disadvantages that are avoided by using REXXRUN.

One way is to use the system CALL command. This has the disadvantage that it requires the system command handler (CMD.EXE), which means the new program must run in a separate process if the calling program wasn't started by CMD.EXE. There is an additional disadvantage that only one argument string can be passed this way.

Another way is to use the REXX CALL statement. This has the disadvantage that it cannot take a variable file name (without using INTERPRET). In addition, if the called program fails because of a REXX error, a SYNTAX condition will be raised in the calling program.

A final way is to use the REXX macrospace. One possible disadvantage of this is that the macrospace is global to the system, raising possibilities of name conflicts.

REXXTHREAD(type, source[, arg-1],...[, arg-n])

Returns the thread id of a newly created thread, or 0 if the thread could not be started.

Type is the type of source code, which can be:

F - Source code is in a file.
S - Source code is in a string.
T - Source code is in a tokenized string.
M - Source code is in the macrospace.

Source depends on the value of *type*. For each type below, *source* can be:

F - the actual program code
S - source code in a string
T - tokenized source code
M - the name of a function in the macrospace

Arg-i are the arguments to be passed to the program.

Source code may be supplied in a REXX string (type 'S'). This should be in a form just as if the program were read from a file, with lines separated by carriage returns and line feeds, and statements on the same line separated by semicolons.

Tokenized code is created as the output of the TOKENIZEFILE or TOKENIZESTRING functions.

Programs are loaded into the macrospace with the MACROADD or MACROLOAD functions.

Because the new program is run on a different thread there is no direct way to access a value returned by the program. Any exchange of data between threads mus use some form of inter-program communication such a REXX external data queues or named pipes.

SCRBLINK([state])

Returns the current state of video attribute handlin (blink flag) and optionally changes it to *state*. *State* ca be either 1 (blink) or 0 (high intensity backgroun colors). Appendix C, on page 214, contains a list of th video attributes.

SCRBORDER([color])

Returns the existing color attribute of the screen borde and uses *color* to establish a new screen border attribute *Color* can be:

0 -	black	4 -	red
1 -	blue	5 -	magenta
2 -	green	6 -	brown
3 -	cyan	7 -	white

SCRCLEAR([attr][, char][, row][, column] →
→ [, height][, weight]

Returns 0 after clearing the entire screen, or an rectangular portion of the screen. Similar in function t SysCls (page 39).

The display *attr*, with a default of 7, is used to clear th area.

The *char*, with a default of blank, is used to clear the area.

The top *row*, with a default of 1, of the area to be cleared.

The left-most *column*, with a default of 1, of the area to be cleared.

The *height*, in rows, of the area to be cleared. Default is to the bottom of the screen.

The *width*, in columns, of the area to be cleared. The default is the right edge of the screen.

Appendix C, on page 214, contains a list of the video attributes.

```
SCROLLDOWN( n[, pad][, attr] →
              → [, top][, left][, bottom][, right] )
```

Returns 1 and scrolls text downward for *n* rows in a rectangular region of the screen defined by *top*, *left*, *bottom*, and *right*. *Pad* (default blank) indicates the character, along with *attr* (default white on black), to be applied to the rows inserted into the top of the rectangular region. Appendix C, on page 214, contains a list of the video attributes.

```
SCROLLLEFT( n[, pad][, attr] →
              → [, top][, left][, bottom][, right] )
```

Returns 1 and scrolls text left for *n* columns in a rectangular region of the screen defined by *top*, *left*, *bottom*, and *right*. *Pad* (default blank) indicates the character, along with *attr* (default white on black), to be applied to the rows inserted into the right portion of the rectangular region. Appendix C, on page 214, contains a list of the video attributes.

SCROLLRIGHT(n[, pad][, attr] →
→ [, top][, left][, bottom][, right])

Returns 1 and scrolls text right for *n* columns in a rectangular region of the screen defined by *top*, *left*, *bottom*, and *right*. *Pad* (default blank) indicates the character, along with *attr* (default white on black), to be applied to the rows inserted into the left portion of the rectangular region. Appendix C, on page 214, contains a list of the video attributes.

SCROLLUP(n[, pad][, attr] →
→ [, top][, left][, bottom][, right])

Returns 1 and scrolls text upward for *n* rows in a rectangular region of the screen defined by *top*, *left*, *bottom*, and *right*. *Pad* (default blank) indicates the character, along with *attr* (default white on black), to be applied to the rows inserted into the bottom of the rectangular region. Appendix C, on page 214, contains a list of the video attributes.

SCRPUT(row, col, string[, option])

Returns a null string after writing *string* to the screen at the position indicated by *row* and *col*. *Option* is Attributes, Text, or Both and indicates the contents of *string*. Appendix C, on page 214, contains a list of the video attributes.

SCRREAD(row, col, length[, option])

A string read from the screen at the position specified by *row* and *col* for a length *length*. *Option* is Attributes, Text, or Both and indicates what is read from the screen. If attributes are included, the length of the string returned is double the number of bytes returned when attributes are not included. Appendix C, on page 214, contains a list of the video attributes.

SCRSIZE()

Returns two words indicating the size of the screen (or window) in rows and columns. Functionally equivalent to SysTextScreenSize() (page 58).

SCRWRITE([row][, col][, string][, length] →
→ [, pad][, attr])

Returns 0 after writing *string*, with optional *attr*, beginning at *row* and *col* for *length* or the length of *string* to the screen. If *length* exceeds the length of string, *pad* is used to complete the write. SCRWRITE does not affect the current cursor position. Appendix C, on page 214, contains a list of the video attributes.

SHIFTSTATE(key,[state])

Returns the current shift state of the NumLock, CapsLock, and ScrollLock keys. The presence of *state* results in the state of the respective key being set.

Key:	'C' (CapsLock); 'N' (NumLock); or 'S' (ScrollLock).
State:	0 to for unshifted, 1 for shifted.

SIN(angle)

Returns the sine of *angle* (expressed in radians). °

SINH(angle)

Returns the hyperbolic sine of *angle* (expressed in radians). °

SOUND([frequency], [duration])

Sounds the computer's speaker. *Frequency*, in cycles per second (Hertz) with a range of 37 to 32,767 and a default of 880, is rounded to the nearest integer. *Duration* is in fractions of a second with a default of 0.2 and accuracy to within about one tenth of a second. Functionally equivalent to BEEP() (page 17).

SQRT(x)

Returns the square root of *x*, a non-negative real number where $x \geq 0$; otherwise, returns 'NAN' (Not A Number). °

STRINGCRC(string)

Returns the 32-bit CRC of *string* as an 8-digit hexadecimal number.

```
Example:     '352441C2' = STRINGCRC( 'abc' )
```

Note: *A CRC (Cyclic Redundancy Check) is a number computed by performing a calculation involving* *every byte of a string. The calculation is designed to minimize the probability that two strings which differ by even one byte can have the same CRC.*

STRINGIN([row][, column][, string] →
→ [, length][, pad][, attr])

Returns the *string* entered into the input field, up to but not including the first *pad* character. The input operation is terminated by pressing the Enter key or by pressing the Esc key in which case a null string is returned.

Row -	Screen row number of the input field (default is row containing cursor).
Column -	Screen column number of the start of the input field (default is the column containing the cursor).
String -	String used as initial contents of input field (default is blanks).
Length -	Length of *string* (default is right edge of the screen).
Pad -	Character used to fill out the input field beyond the entered data (default is blank).
Attr -	Attribute to be used for displaying characters entered in the field (default is 7, white on black). Appendix C, on page 214, contains a list of the video attributes.

TAN(angle)

Returns the tangent of *angle* (expressed in radians). °

TANH(angle)

Returns the hyperbolic tangent of *angle* (expressed in radians). °

TOKENIZEFILE(file_name, output)

Returns 1 indicating that the REXX language processor successfully produced a tokenized program string; otherwise, returns 0.

File_name is the name of a REXX program file.

Output is the name of a REXX variable that receives the tokenized result.

A tokenized program string can be used by the REXXRUN() and REXXTHREAD() functions to execute a program without the overhead of processing the source code for each call.

rOKENIZESTRING(source, output)

Returns 1 indicating that the REXX language processor successfully produced a tokenized program string; otherwise, returns 0.

Source is a REXX string containing program source code.

Output is the name of a REXX variable that receives the tokenized result.

A tokenized program string can be used by the REXXRUN and REXXTHREAD functions to execute a program without the overhead of processing the source code for each call.

Tokenized code is language processor dependent. Code tokenized by Personal REXX cannot be used by IBM REXX and vice versa.

The source string should be in a form just as if the program were read from a file, with lines separated by carriage returns and line feeds, and statements on the same line separated by semicolons.

rYPEMATIC(rate, delay)

Returns 1 indicating that the typematic repeat *rate* (characters per second - typically < 30) and initial *delay* (time in milliseconds - typically < 1000) were successfully set; otherwise, returns 0.

If the function is invoked in a PM window, it returns 0 indicating no change; however, if it is invoked in a full-screen session, it applies to all active sessions, including PM sessions.

JPPER(string)

Returns *string* in uppercase.

VALIDNAME(file_name[, wildcard])

Returns 1 if *file_name* is a syntactically valid file name. *Wildcard* is 1 (wild card characters are permitted) or 0 (no wild card characters are permitted) in *file_name*.

VARDUMP([file_name][, [include][, var1],] ...)

Returns 1 indicating that the selected variables have been written to the fully qualified *file_name* (default STDOUT). *Include* can be 'I' or 'E', to include or exclude respectively, the variables named in *var1* - *varn*.

If *include* is 'I' and no variables are named, all variables are written to *file_name*. Output will be appended to *file_name* if it exists.

This is intended as a debugging function.

VARREAD(file_name[, [include][, var1],] ...)

Returns 1 indicating that the selected variables have been read from the fully qualified *file_name*. *Include* can be 'I' or 'E', to include or exclude respectively, the variables named in *var1* - *varn*.

If *include* is 'I' and no variables are named, all variables are read from file_name.

VARWRITE(file_name[, [include][, var1],] ...)

Returns 1 indicating that the selected variables have been written to the fully qualified *file_name*. *Include* can be 'I' or 'E', to include or exclude respectively, the variables named in *var1* - *varn*.

If *include* is 'I' and no variables are named, all variables are written to *file_name*. If *file_name* exists, it should have been created by VARWRITE and will be appended to.

3.2 RXWINDOW Functions

RXWINDOW can be registered with:

```
    call RxFuncAdd 'W_Register', 'RXWIN30', 'rxwindow'
    call W_Register
```

It can be removed (unregistered) with:

```
    call W_Deregister
```

W_ATTR(window, row, col, length, attr)

Returns 1 after setting the screen attributes of a portion of a line in *window* (the handle returned by W_OPEN).

Row and *col* are the row and column numbers respectively within the window where *attr* display attributes are to take effect on subsequent output. The default for *attr* is defined by the call to W_OPEN.

W_BORDER(window[, top][, right][, bottom][, left] →
→ [, attr])

Returns 1 after displaying a border for *window* (the handle returned by W_OPEN).

Each side of the window can be individually defined with the value 0 (no lines), 1 (a single line), or 2 (double lines) in *top*, *right*, *bottom*, and *left*.

Attr is the display attribute to be used for the border, with the default being defined by the call to W_OPEN.

W_CLEAR(window[, attr][, char] →
→ [, row][, col][, height][, width])

Returns 1 after clearing *window* (the handle returned by W_OPEN) or a rectangular portion of *window*. *Attr* and *char* can be used to fill the cleared area.

Height and *width* define the respective sizes of the area to be cleared, each with a default of the remaining area on the screen. W_CLEAR() is analogous to the SCRCLEAR() function (page 92).

W_CLOSE(window)

Returns 1 after permanently closing *window* (the handle returned by W_OPEN) and removing all related information. The screen contents under the window are restored to what they were before *window* was opened.

W_FIELD(window, field_name, row, col, length →
→ [, attr][, pad])

Returns 1 indicating that a named area, *field_name*, has been defined within *window* (the handle returned by W_OPEN) as an input area where data can be keyed; otherwise, returns 0 if *field_name* would not be positioned within *window*.

Field_name is the name assigned to the field.

Row and *col* contain the respective positions in *window* where the field begins, with 1, 1 representing the upper left corner of *window*. *Length* contains the number of character positions occupied by *field_name* and is truncated at the end of the specified row in *window* and does not wrap around.

Attr is the display attribute to be used for the field, with the default being defined by the call to W_OPEN. *Pad*, with a default of blank, is the character used to fill the area.

W_GET(window, row, col, length[, string] →
→ [,attr][, pad][, activate])

Returns data keyed into the area in *window* (the handle returned by W_OPEN) defined by *row* and *col* with a size of *length* without wrap around.

String defines the initial contents of the area and is returned if no data is keyed into the area.

Attr is the display attribute to be used for the field, with the default being defined by the call to W_OPEN. *Pad*, with a default of blank, is the character used to fill the area. Pad characters at the end of the entered data, and only at the end, are stripped before the data is returned.

Activate instructs W_GET how the input operation is deemed complete.

N (Normal) indicates that only the Enter and Esc keys result in the input operation completing. The Enter key results in the keyed data being returned while the Esc key results in a null string being returned.

F (Function keys) allows the Enter and the Esc keys along with other keys, except for the editing keys listed below, to result in the input operation completing with the keyed data being returned (even as a result of the Escape key).

The name of the key that resulted in the input operation completing will be placed in the REXX special variable *_ACTIVATION_KEYNAME* and can be any of the following:

```
ENTER                      C-F1 ... C-F12
ESC                        A-F1 ... A-F12
F1 ... F12                 PGUP, PGDN
S-F1 ... S-F12             OTHER
```

W_HIDE(window[, option])

Returns 1 and temporarily inhibits the display of further data written to window (the handle returned by W_OPEN) and removes the window from the screen, making whatever was behind the window visible.

Option can be:

A All, the default, which causes all data in the window to become invisible.

N New, which results in only new data written to *window* to be invisible.

W_ISFIELD(window, field_name)

Returns 1 if *field_name* is defined within *window* (the handle returned by W_OPEN); otherwise, returns 0.

W_ISWINDOW(window)

Returns if *window* (the handle returned by W_OPEN) represents a valid, currently open window; otherwise, returns 0.

W_KEYS(window[, tab_option][, enter-option] →
→ [, keyboard-option])

Returns 1 and controls various aspects of cursor and keyboard operations with W_GET and W_READ functions for *window* (the handle returned by W_OPEN).

Tab_option can be:

J Jump, which causes the cursor to automatically jump from one field to the next when W_READ is used with multiple fields.

<u>N</u> No jump, the default, which does not allow the cursor to automatically jump to the next field.

Enter_option affects the action resulting from pressing the Enter key and can be:

<u>A</u> Any field, the default, causes the Enter key to act as an activation key when the cursor is in any field.

L Last field, causes the Enter key to act as an activation key only when it is used in the last field of a window (as defined by the order of W_FIELD calls). When the cursor is in any other, the Enter key causes it to jump to the next field.

Keyboard-option controls the action of keys on an IBM Enhanced Keyboard and can be:

E Enhanced mode which allows F11, F12 and dedicated key pad keys to act as activation keys.

<u>F</u> Fold mode, the default, causes keyboard scan codes to be "folded" thus, with an Enhanced Keyboard, analogous keys on the numeric and dedicated key pads return the same codes.

W_MOVE(window, row, col)

Returns 1 if *window* (the handle returned by W_OPEN) can be successfully moved such that the new top, left position of the window occupies the position defined by *row* and *col*; otherwise, returns 0 indicating that no move took place because the resulting position of *window* would be off of the screen.

Row and *col* represent values relative to the entire screen.

If the move is successful, data beneath the window which is now uncovered becomes visible.

W_OPEN(row, col, height, width[, attr])

Returns the "handle" of a new window opened with the described characteristics; otherwise, a null string is returned if the open is unsuccessful. This handle is then used as the first parameter in all of the other RXWINDOW functions. Up to 5000 windows can be open at one time.

Row and *col* specify the position of the top left corner of the window on the screen with 1, 1 being the top left corner of the screen.

Height and *width* represent the number of rows and columns respectively the window is to occupy. The limit for each of these fields is the size of the screen as returned by any of the functions that return screen size (SCRSIZE() - page 94, SysTextScreenSize() - page 58, etc.).

Attr, with a default of 7 (white on black), indicates the attribute which will be used for all other RXWINDOW functions when a new attribute is not explicitly indicated for that function. A complete list of the video attributes will be found in Appendix C (page 214).

W_PUT(window, row, col[, string][, length] →
→ [, attr][, pad])

Returns 1 after displaying *string* for *length* characters, padded with *pad*, at *row* and *col* within *window* (the handle returned by W_OPEN). If *string* is omitted, a null string is written.

W_PUT is included primarily for compatibility with a prior version of RXWINDOW. See the *REXXLIB User's Guide* for further details.

W_READ(window, [field_name][, activate])

Returns 0 if *activate* is 'N' and the read is terminated with the Esc key; otherwise, returns 0. W_READ reads user input from any currently defined input fields in *window* (the handle returned by W_OPEN). At least one such field must be defined.

If *field_name* is specified, it designates the field in which the cursor will initially be positioned. Alphabetic case is ignored. If *field_name* is not specified, the cursor will initially be placed at the beginning of the first input field defined for the window.

Activate instructs W_READ how the input operation is deemed complete.

N (Normal) indicates that only the Enter and Esc keys result in the input operation completing. The Enter key results in the keyed data being returned while the Esc key results in a null string being returned.

F (Function keys) allows the Enter and the Esc keys along with other keys (except for the editing keys referenced in connection with W_GET() - page 100), to result in the input operation completing with the keyed data being returned (even as a result of the Escape key).

In addition to the field variables, the following special variables are set:

- *_ACTIVATION_KEY* contains the code for the key that ended W_READ, if *activate* is 'F' (see Appendix B - page 211).
- *_ACTIVATION_KEYNAME* contains the name, in uppercase, of the key that ended W_READ, if *activate* is 'F' (see W_GET - page 100).
- *_ACTIVATION_FIELD* contains the name, in uppercase, of the field that the cursor was in when

the function terminated, unless *activate* is 'N' and the Esc key was pressed.

W_SCRPUT(window, row, col, string[, option])
Returns 1 after displaying a string of text, attributes, or both in *window* (the handle returned by W_OPEN).

Row and *col* represent positions within *window*, with the top, left corner of the window being 1, 1.

Option can be:

T Text, the default, indicates that *string* contains only displayable characters. Attributes already present do not change.

A Attributes only. Text already present changes in appearance only.

B Both text characters and attributes are included in *string* as character-attribute pairs with the attribute byte following its text character.

W_SCRPUT is analogous to the SCRPUT() function (page 94).

W_SCRREAD(window, row, col, length[, option])
Reads and returns a string of text, attributes, or both from *window* (the handle returned by W_OPEN). *Row* and *col*, (with the top, left position of the window being 1,1) indicate the position in the window where reading begins.

Length is the number of screen positions in the window to read. If the data to be read extends beyond the end of a line, the read will wrap around, for as many lines as necessary, up to the end of the screen.

Option can be:

T Text, the default, indicates that only text characters will be returned.

A Attributes only will be read.

B Both text characters and attributes are included ir the returned string as character-attribute pairs witl the attribute byte following its text character.

Only data which has been written to the window witl RXWINDOW functions is read by W_SCRREAD W_SCRREAD() is analogous to the SCRREAD(function (page 94).

W_SCRWRITE(window, row, col[, string][, length] →
→ [, pad][, attr])

Returns 1 after writing *string*, with optional *attr* beginning at *row* and *col* for *length* or the length of *strin* to *window* (the handle returned by W_OPEN).

Row and *col* represent a position within the window witl the top, left corner of the window being 1, 1.

If *length* exceeds the length of string, *pad* is used tc complete the write. Appendix C, on page 214, contain a list of the video attributes.

W_SIZE(window)

Returns two words indicating the size of the window ir rows and columns.° Analogous to SysTextScreenSize((page 58) and SCRSIZE() (page 94).

W_UNFIELD(window, field_name)

Returns 1 and removes *field_name* from *window* (the handle returned by W_OPEN). *Field_name* must have previously been defined with W_FIELD. The area of the window occupied by *field_name* will be cleared to blank with the default window attribute. It is not necessary tc issue W_UNFIELD() prior to issuing W_CLOSE() tc close a window.

W_UNHIDE(window)

Returns 1 after reversing the effect of W_HIDE making *window* (the handle returned from W_OPEN) and al further updates to it fully visible.

4. Workplace Shell

The Workplace Shell information has been collected from many different sources. Some of it was provided by, and included with the permission of Development Technologies, Inc. Their information was collected during the development of, and through the use of, their DeskMan/2 product.

4.1 WPS Objects

The table below (in alphabetic sequence) contains most, though not necessarily all, of the predefined WPS object IDs and the functions they belong to. The title field contains the data extracted from the actual object. Where the title appears on more than one line in the table, the actual title contains a single space at the end of each table line. The carat character (^) implies a new line in the title beneath the object's icon.

The list includes the object IDs created by a complete custom installation of OS/2 Warp 4. Please note that object IDs are case sensitive. For example, you cannot specify <WP_DESKTOP> & <WP_Desktop> interchangeably. Many of the object titles were changed in Warp Version 4. Where a different title is assigned to the same object ID, both are shown. Notations within parenthesis are descriptive only.

Object ID	Title	
<16Color_Template>	Solid Color Palette	(V3)
<256Color_Template>	Mixed Color Palette	(V3)
<ADV_ASSIST>	Customer^Assistance	(V3)
<ADV_ASSIST_REG>	Registration	(V3)
<ADV_DIALER>	IBM Internet^Dialer	(V3)
<ADV_REG>	IBM Internet^Customer Services	(V3)
<ADV_REG>	IBM Internet^Customer Services^(Modem)	(V4)
<AOS2WARP>	AskPSP	(V4)
<ASKREADME>	AskPSP^ReadMe	(V4)
<AVI_FILE_UTILITY>	AVI File Utility	(V4)
<Address^Book>	Address^Book	(V3)
<CAGINSTS>	SystemView Agent^Remove	(V4)
<CASOS2>	OS/2 agent startup	(V4)

Object ID	Title
<CLTRMOVE>	Remove Installation^for Networking (V4)
<CLTSTART>	Selective Install^for Networking (V4)
<CLT_REMOTE>	OS/2 Warp^Remote Install (V4)
<CPREADME>	CasePoint^Release Notes (V4)
<Call^Manager>	Call^Manager (V3)
<Chalkboard>	Chalkboard (V3)
<Clip>	Clip (V3)
<FFST_SETUP>	FFST Setup (V4)
<FPGuide>	File and Print Client^Guide (V4)
<FPW_CATALYST>	Footprint Catalyst (V3)
<FPW_HALITE>	HyperACCESS^Lite (V3)
<FTPPM Template>	FTP-PM (V4)
<Getting^Started>	Getting^Started (V3)
<IAK>	IBM Internet^Connection for OS/2 (V3)
<IAK>	Internet^(Modem) (V4)
<IAK_3270>	3270 Telnet (V3)
<IAK_FTP>	FTP-PM (V3)
<IAK_FTP>	FTP-PM^(Modem) (V4)
<IAK_GOPHER>	Gopher (V3)
<IAK_GOPHER>	Gopher^(Modem) (V4)
<IAK_HELP>	Introduction to the^IBM Internet Connection (V3)
<IAK_NR2>	NewsReader/2 (V3)
<IAK_NR2>	NewsReader/2^(Modem) (V4)
<IAK_README>	READ ME FIRST (V3)
<IAK_RSU>	Software Updates (V4)
<IAK_SLIPPM>	Dial Other^Internet Providers (V3)
<IAK_TELNET>	Telnet (V3)
<IAK_TELNET>	Telnet^(Modem) (V4)
<IAK_TEMPLATES>	Application^Templates(V3)
<IAK_TEMPLATES_3270>	3270 Telnet (V3)
<IAK_TEMPLATES_FTPPM>	FTP-PM (V3)
<IAK_TEMPLATES_FTPPM>	FTP-PM^Modem (V4)
<IAK_TEMPLATES_HELP>	How do I^use Templates? (V3)
<IAK_TEMPLATES_TELNET>	Telnet (V3)
<IAK_TEMPLATES_TELNET>	Telnet^Modem (V4)

Object ID	Title
<IAK_UPDATE>	Retrieve^Software Updates (V3)
<IAK_UTILITIES>	Internet^Utilities (V3)
<IAK_UTILITIES>	Internet Utilities^(Modem) (V4)
<IAK_WEB>	WebExplorer^(Modem) (V4)
<IAK_WEB_SHAD>	WebExplorer^(Modem) (V4)
<InfoHighway>	IBM Information^Superhighway (V3)
<JAVA_Copyright>	copyrght (V4)
<JAVA_EditorforJava>	Editor for Java (V4)
<JAVA_JavaAppletViewerFromHTML>	Java Applet Viewer from HTML (V4)
<JAVA_JavaAppletViewerFromURL>	Java Applet Viewer from URL (V4)
<JAVA_JavaCompiler>	Compile Java code (V4)
<JAVA_JavaDisassemble>	Disassemble Java class file (V4)
<JAVA_JavaDispMethod>	Display methods in Java class file (V4)
<JAVA_JavaGenDoc>	Generate documentation from Java code (V4)
<JAVA_JavaWindow>	OS/2 Window for Java(V4)
<JAVA_OS2>	Java for OS/2 (V4)
<JAVA_OS2_SMPLS>	Samples for Sun's Java Programming Environment (V4)
<JAVA_OS2_SMPLS_URLS>	URLs for Samples (V4)
<JAVA_OS2_TLKT>	Toolkit for Sun's Java Programming Environment (V4)
<JAVA_RunJavaPMProgram>	Run Java PM Program (V4)
<JAVA_RunJavaProgram>	Run Java Program (V4)
<JAVA_SHDW_demo>	demo (V4)
<JAVA_SHDW_weblogs>	weblogs (V4)
<KARATCA>	SystemView Agent^System Management Agent (V4)
<LDCS_REMOVE>	Remote Access Client^Remove (V4)

Object ID	Title
<LD_ADV_GUIDE>	Remote Access Advanced Guide^LAN Distance Advanced Guide (V4)
<LD_CLIENT_GUIDE>	Remote Access Client^Guide (V4)
<LSGuide>	LAN Administration^Guide (V4)
<LS_ADMIN>	LAN Server^Administration (V4)
<LS_ADMIN_SHADOW>	LAN Server^Administration (V4)
<LS_AUDIT>	Audit Log (V4)
<LS_CLIP>	Network DDE^and Clipboard (V4)
<LS_ERROR>	Error Log (V4)
<LS_FOLDER>	Logons (V4)
<LS_FOLDER_SHADOW>	Logons (V4)
<LS_INSTALL>	File and Print Client^Install/Remove (V4)
<LS_NETMSG>	Network Messaging (V4)
<LS_START>	Start^File and Print Client (V4)
<LT_SAMPLE>	Multimedia Viewer (V3)
<MAH_EXE>	Mahjongg (V3)
<MAH_FOLDER>	Mahjongg Solitaire (V3)
<MAH_Folder>	Mahjongg Folder (pre V3)
<MARSCM1>	Mobile Office Services (V4)
<MCIREXX_INF>	Multimedia With REXX
<MCMDOC>	Mobile Office Services^Guide (V4)
<MCMPROG>	Start^Mobile File Sync^Mobile Office Services (V4)
<MCMSPY>	Spy Utility (V4)
<MCMSTOP>	Stop^Mobile File Sync^Mobile Office Services (V4)
<MFSCP>	Clear Persistence (V4)
<MFSREADME>	MFS README (V4)
<MFSS>	MFS Setup (V4)

Object ID	Title
<MFSST>	Stashing Database (V4)
<MMPM_CDPLAYER1>	Compact Disc (V3)
<MMPM_DAPLAYER1>	Digital Audio (V3)
<MMPM_MIDIPLAYER1>	MIDI (V3)
<MMPM_VIDPLAYER1>	Videodisc (V3)
<MMPM2_AUDIOFINDER>	Digital audio (V4)
<MMPM2_AVI_FILE_UTILITY>	AVI File Utility (V3)
<MMPM2_BITMAP>	Bitmaps (V4)
<MMPM2_FOLDER>	Multimedia (V4)
<MMPM2_Folder>	Multimedia *(Folder)* (V3)
<MMPM2_IMAGEFINDER>	Image (V4)
<MMPM2_IMAGES>	Images (V4)
<MMPM2_MASTERVOLUME>	Volume Control *(Desktop)*
<MMPM2_MASTERVOLUME_D>	Volume Control *(MMPM2 Folder)*
<MMPM2_MIDIFINDER>	MIDI (V4)
<MMPM2_MINSTALL>	Multimedia Install (pre V3)
<MMPM2_MINSTALL>	Multimedia^Application Install (V4)
<MMPM2_MMCDDEVICE01>	Compact Disc (V4)
<MMPM2_MMCDDEVICETEMPLATE>	Compact Disc (V4)
<MMPM2_MMCONVERTER>	Multimedia Data Converter
<MMPM2_MMFOLDERTEMPLATE>	Lighttable (V4)
<MMPM2_MMLVDDEVICETEMPLATE>	Videodisc (V4)
<MMPM2_MMTEMPLATEFOLDER>	Multimedia Templates (V4)
<MMPM2_MMVOLUME>	Volume (V4)
<MMPM2_MOVIES>	Movies
<MMPM2_README>	README
<MMPM2_SETUP>	Multimedia Setup
<MMPM2_SNDSHAD>	Sound *(Sound Shadow)*
<MMPM2_SOFTWARE_MOTION_VIDEO1>	Digital Video
<MMPM2_SOFTWARE_VIDEO_RECORDER>	Video IN^Recorder (V3)
<MMPM2_SOUNDS>	Sound Bites
<MMPM2_VIDEOFINDER>	Digital video (V4)
<MPTSCfg>	Network Adapters and^Protocol Services Guide (V4)
<MPTS_DDNS_OBJECT_SETUP>	DDNS^Configuration (V4)
<NFCLIENTGUIDE>	System Management^Client Guide (V4)

Object ID	Title
<NFNETDRV>	Network Driver Configuration (V4)
<NFNETIPC>	TME 10 NetFinity Network Interface (V4)
<NFREADME>	TME 10 NetFinity^System Management Client^Read Me (V4)
<NFREADME_SHAD>	TME 10 NetFinity^System Management Client^Read Me (V4)
<NFSVCMGR>	TME 10 NetFinity Service Manager(V4)
<NP_DHCP_GUIDE>	TCP/IP DHCP Client^Administration Guide (V4)
<NP_DIP_INTRO>	TCP/IP Dynamic IP^Introduction (V4)
<NSC_FOLDER>	Password^Coordination (V4)
<NSC_PM>	Password^Coordination (V4)
<NSC_REF>	Password Coordination^Guide (V4)
<NSC_REMOVE>	Password Coordination^Remove (V4)
<NSC_RETRY>	Password Coordination^Retry Process (V4)
<NSC_SVR>	Password Coordination^Server (V4)
<NTS_MPTS_ICON>	MPTS^Network Adapters^and Protocol Services (V4)
<ODOC_FOLDER>	OpenDoc for OS/2 (V4)
<OD_INTRO>	OpenDoc Guide (V4)
<OD_PREFS>	Part Editor Preferences (V4)
<OD_SCRPER>	Script Editor^and Recorder (V4)
<OD_SHELLPLUGINS>	OpenDoc Shell Plug-ins (V4)
<OD_TEMPS>	OpenDoc Templates (V4)
<P2P/2>	P2P/2 (V3)
<PCOMADM>	Administration Tools (V4)

Object ID	Title
<PCOMOS2>	Personal Communications^3270/5250 Emulation (V4)
<PDP_FOLDER>	Problem Determination Tools (V4)
<PDP_PMDF>	PM Dump Facility (V4)
<PDP_PMDFDOC>	PMDF Doc (V4)
<PDP_SMARTPFA>	Hard File Monitor (V4)
<PDP_SYSLOG>	System Error Log (V4)
<PDP_TRACE_DOC>	Trace Doc (V4)
<PDP_TRACE_FMT>	Trace Formatter (V4)
<PDP_TRACE_FOLDER>	Trace Options (V4)
<PDP_TRACE_OFF>	Trace OFF (V4)
<PDP_TRACE_OFFCLEAR>	Trace OFF Clear Buffer (V4)
<PDP_TRACE_ON>	Trace ON (V4)
<PDP_TRACE_ONCLEAR>	Trace ON Clear Buffer (V4)
<PDP_TRACE_ONRESUME>	Resume Trace (V4)
<PDP_TRACE_ONSUSPEND>	Suspend Trace (V4)
<PEER_LANLOGON>	LAN Server^Logon (V4)
<PEER_LOCALLOGON>	File and Print Client^Workstation Logon (V4)
<PEER_LOGOFF>	Logoff (V4)
<PEER_USER>	Network User Account (V4)
<PEER_WKST>	Shared Resources and^Network Connections (V4)
<PMDM2IMAGE>	PM Image
<Pupil731B>	Remote Support^for OS/2 (V4)
<Pupil731B_Disk>	Remote Support^for OS/2^Create Client diskette in A: (V4)
<Pupil731B_Guide>	Remote Support for OS/2^User Guide (V4)
<Pupil731B_Info>	Remote Support for OS/2^User Guide (V4)
<README_MM>	Double-Click Here *(Multimedia Readme)* (V3)
<RPSM_FLD>	TME 10 NetFinity^System Management Client (V4)
<SAMPLE_SOURCE>	Source (V4)

Object ID	Title
<SNMPCNFIG>	OS/2 Agent Configuration (V4)
<SNMPTRAP>	SNMPTRAP (V4)
<SPCH_CHKINST>	Check Installation (V4)
<SPCH_DICTATION>	Dictation Window (V4)
<SPCH_ENROLL>	Enrollment (V4)
<SPCH_GAMES>	States Game (V4)
<SPCH_INSTALLILM>	Optional Vocabularies Install/Uninstall (V4)
<SPCH_MACROEDIT>	Dictation Macro Editor (V4)
<SPCH_MIGRATE>	Migrate User Information (V4)
<SPCH_USER>	Speech User (V4)
<SPCH_USERGUIDE>	VoiceType User's Guide (V4)
<SPCH_WW>	Voice Manager (V4)
<SVAUSR>	System Management^Agent Guide (V4)
<SVA_README>	IBM SystemView Agent Readme (V4)
<SVA_README_SHAD>	IBM SystemView Agent Readme (V4)
<SampleApp>	CompuServe (V3)
<SampleApp	EXE> CIM for OS/2 (V3)
<SampleApp	ReadMe> Read Me (V3)
<SampleApp	Signup> Member Signup(V3)
<Scheme_Palette_96_Template>	Scheme Palette (V4)
<Stills^Capture>	Stills^Capture (V3)
<TCP/IP15>	TFTP (V4)
<TCP/IP15_SHAD>	TFTP (V4)
<TCP/IP17>	TCP/IP^Command Reference (V4)
<TCP/IP18>	TCP/IP^Guide (V4)
<TCP/IP1>	FTP (V4)
<TCP/IP1_SHAD>	FTP (V4)
<TCP/IP50>	TCP/IP Startup (V4)
<TCP/IP5>	PM Ping (V4)
<TCP/IP5_SHAD>	PM Ping (V4)
<TCP/IP8>	TCP/IP^Configuration → ^(LAN) (V4)
<TCP/IP8_SHAD>	TCP/IP^Configuration → ^(LAN) (V4)

Object ID	Title
<TCP/IP>	TCP/IP^Internet^(LAN) (V4)
<TCP/IPDB1>	Windows-Ping (V4)
<TCP/IPDB1M>	Windows-Ping (V4)
<TCP/IPDB2>	DOS Ping (V4)
<TCP/IPDB2M>	DOS Ping (V4)
<TCP/IP_GOPHER>	Gopher^(LAN) (V4)
<TCP/IP_NR>	NewsReader/2^(LAN) (V4)
<TCP/IP_README_DBX1>	TCP/IP for ^DOS/Windows^Read Me (V4)
<TCP/SETTERM>	Telnet^Customization (V4)
<TCP/SETTERM_SHAD>	Telnet^Customization (V4)
<TCPFTP>	FTP-PM^LAN (V4)
<TCPFTPRX>	TCP/IP REXX FTP → API^Reference (V4)
<TCPIP_WEB>	WebExplorer^(LAN) (V4)
<TCPIP_WEB_SHAD>	WebExplorer^(LAN) (V4)
<TCPSORX>	TCP/IP REXX Sockets API^Reference (V4)
<TCPTELNET>	Telnet^LAN (V4)
<TCP_FTP>	FTP-PM^(LAN) (V4)
<TCP_README_INF>	TCP/IP^Readme (V4)
<TCP_TELNET>	TelnetPM^(LAN) (V4)
<TCP_UINSTALL>	TCP/IP Services^Remove (V4)
<TCP_UTIL_FOLDER>	TCP/IP Utilities^(LAN) (V4)
<TCP_UTIL_FOLDER_SHAD>	TCP/IP Utilities^(LAN) (V4)
<TK_CSAMPLE>	Sample Programs Folder
<TK_DEVINFO>	Toolkit Information Folder
<TK_DEVTOOLS>	Development Tools Folder
<TK_TOOLKIT>	Toolkit 2.1 Folder
<Talk>	Talk (V3)
<ULTIMAIL_ADDRBOOK>	Names and^Addresses
<ULTIMAIL_CABINET>	Mail Cabinet
<ULTIMAIL_INBASKET>	In-basket
<ULTIMAIL_INFO_LITE>	Information (V3)
<ULTIMAIL_LITE>	Ultimedia^Mail/2 'Lite' (V3)
<ULTIMAIL_NEWLETTR>	New Letter
<ULTIMAIL_SHADOW>	Ultimedia^Mail/2 'Lite' (V4)

Object ID	Title
<UMAIL_VIEW_FAQ_LITE>	UltiMail 'Lite'^Frequently asked^Questions (V4)
<UMAIL_VIEW_README_LITE>	Read Me (V3)
<UMAIL_VIEW_README_LITE>	UltiMail 'Lite'^Read Me (V4)
<UMAIL_VIEW_TUTORIAL_LITE>	Tutorial (V3)
<UMAIL_VIEW_USG_LITE>	User's Guide (V3)
<UMAIL_VIEW_USG_LITE>	UltiMail 'Lite'^Guide (V4)
<UPM_ACCTS>	User Account → Management (V4)
<UPM_FOLDER>	UPM Services (V4)
<UPM_LOGOFF>	Logoff (V4)
<UPM_LOGON>	Logon (V4)
<URLF_BUS>	Business & Shopping (V4)
<URLF_COMP>	Computing (V4)
<URLF_EDU>	Education (V4)
<URLF_ENT>	Entertainment (V4)
<URLF_IBM>	IBM Web Pages (V4)
<URLF_NEWS>	News and Sports (V4)
<URLF_OS2>	OS/2 Related Web Pages (V4)
<URLF_REF>	Reference (V4)
<URLF_SEARCH>	Web Search Sites (V4)
<VIDEOIN_INFO>	VideoIn Users Guide (V4)
<VIDEOIN_README>	README.VIN (V3)
<VIDEOIN_README>	VideoIN^Readme (V4)
<VIDEO_RECORDER>	VideoIN^Recorder (V4)
<Video>	Video (V3)
<WC_NETSERV>	Network Services (V4)
<WC_WEBEX_FOLD>	WebExplorer (V4)
<WPINET_HOSTTEMPLATE>	FTP HOST (V4)
<WPINET_TEMPLATES>	Templates for^Internet (V4)
<WPINET_URLFOLDERTEMPLATE>	URL Folder (V4)
<WP_APPLBK>	Application Considerations (V3)
<WP_APPSFOLDER>	Applications (V4)
<WP_ART>	Software Registration (V4)
<WP_ASSISTANCE>	Assistance Center (V4)
<WP_BOOT>	Create Utility Diskettes (V3)

Object ID	Title	
<WP_CHART>	PM Chart	(pre V3)
<WP_CHESS>	OS/2 Chess	
<WP_CLIPV>	Clipboard Viewer	
<WP_CLOCK>	System Clock	
<WP_CLRPAL>	Color Palette	(pre V3)
<WP_CMDREF>	OS/2 Warp Command Reference	(V4)
<WP_CNTRY>	Country	
<WP_CONFIG>	System Setup	
<WP_CONNECTIONSFOLDER>	Connections	(V4)
<WP_COOLURLSFOLDER>	Web Sites	(V4)
<WP_DALARM>	Alarms	(pre V3)
<WP_DBASE>	Database	(pre V3)
<WP_DBOOT>	Dual Boot	(pre V3)
<WP_DCALC>	Calculator	(pre V3)
<WP_DCALEM>	Calendar	(pre V3)
<WP_DDARC>	Planner Archive	(pre V3)
<WP_DDIARY>	Daily Planner	(pre V3)
<WP_DDINST>	Device Driver Install	
<WP_DESKTOP>	Desktop	
<WP_DLIST>	Activities List	(pre V3)
<WP_DMNTH>	Monthly Planner	(pre V3)
<WP_DNOTE>	Notepad	(pre V3)
<WP_DOSFS>	DOS Full Screen	
<WP_DOSWIN>	DOS Window	
<WP_DOS_DRV_A>	DOS From Drive A:	
<WP_DOS_DRV_A>	DOS from Drive A:	(V4)
<WP_DRIVES>	Drives	
<WP_DRIVESSHADOW>	Drives	(V4)
<WP_DTARC>	To-Do List Archive	(pre V3)
<WP_EPM>	Enhanced Editor	
<WP_FNTPAL>	Font Palette	
<WP_FPWO_CHTEMP>	Chart	(V3)
<WP_FPWO_DBTEMP>	Database	(V3)
<WP_FPWO_EXE>	IBM Works	(V3)
<WP_FPWO_FOLD>	IBM Works *(folder)*	(V3)
<WP_FPWO_GRTEMP>	Contact List *(template)*	(V3)
<WP_FPWO_README>	ReadMe	(V3)
<WP_FPWO_RWTEMP>	Report *(template)*	(V3)
<WP_FPWO_SAMP_FOLD>	TEMPLATE	(V3)
<WP_FPWO_SSTEMP>	Sheet *(template)*	(V3)
<WP_FPWO_WPTEMP>	Document *(template)*	(V3)

Object ID	Title	
<WP_FPWPIMA_EXE>	Appointments	(V3)
<WP_FPWPIME_EXE>	Event Monitor	(V3)
<WP_FPWPIMG_EXE>	Planner	(V3)
<WP_FPWPIMN_EXE>	Notepad	(V3)
<WP_FPWPIMP_EXE>	Phone/Address Book	(V3)
<WP_FPWPIMS_EXE>	PIM Preferences	(V3)
<WP_FPWPIMT_EXE>	To Do List	(V3)
<WP_FPWPIMY_EXE>	Year Calendar	(V3)
<WP_GAMES>	Games *(folder)*	
<WP_GLOSS>	Glossary	
<WP_HIRESCLRPAL>	Mixed Color Palette	(V3)
<WP_HWMGR>	Hardware Manager	(V4)
<WP_ICON>	Icon Editor	
<WP_IGSCHECKIN>	User Check-In for WarpGuide	(V4)
<WP_IGSFOLDER>	WarpGuide	(V4)
<WP_INFO>	Information	
<WP_INFOOVERFOLDER>	Online Information Overview	(V4)
<WP_INST>	Selective Install	
<WP_INSTALLED>	Installed Features	(V4)
<WP_INSTREMFOLDER>	Install/Remove	(V4)
<WP_JIGSAW>	Jigsaw	(pre V3)
<WP_KEYB>	Keyboard	
<WP_KLDK>	Klondike Solitaire	(V4)
<WP_KLDK>	Solitaire - Klondike	
<WP_LAUNCHPAD>	LaunchPad	(V3)
<WP_LORESCLRPAL>	Solid Color Palette	(V3)
<WP_MIGAPP>	Add Programs	(V4)
<WP_MIGAPP>	Migrate Applications	
<WP_MINDEX>	Help Index	(V4)
<WP_MINDEX>	Master Help Index	
<WP_MOUSE>	Mouse *(Settings)*	
<WP_MULTIMBK>	Multimedia	(V3)
<WP_NEKO>	Cat and Mouse	(pre V3)
<WP_NETWORK>	Network *(folder)*	
<WP_NEWFOLDER>	New Folder	(V4)
<WP_NEWPROGRAM>	New Program	(V4)
<WP_NEWREMOTEPRINTER>	Network Printer	(V4)
<WP_NOWHERE1>	Nowhere *(Hidden folder)*	(V3)
<WP_NOWHERE>	Nowhere *(Hidden folder)*	
<WP_ONLINE>	Using Online Information	(V4)

Object ID	Title
<WP_OS2FS>	OS/2 Full Screen
<WP_OS2SYS>	OS/2 System
<WP_OS2UGBK>	OS/2 Warp Desktop → Guide (V4)
<WP_OS2WIN>	OS/2 Window
<WP_OVERVIEW>	OS/2 Warp System Overview (V4)
<WP_PDVIEW>	Printer (pre V3)
<WP_PERFBK>	Performance Considerations (V3)
<WP_PICV>	Picture Viewer
<WP_POWER>	*(Power Mgmt. - no title specified)* (pre V3)
<WP_PRINTBK>	Printing in OS/2 (V3)
<WP_PRINTERSFOLDER>	Printers (V4)
<WP_PROGRAMSFOLDER>	Programs (V4)
<WP_PROMPTS>	Command Prompts
<WP_PTR_AQUA>	Aqua 3D Pointers (V4)
<WP_PTR_BIG_BLAC>	Big Black Pointers (V4)
<WP_PTR_BIG_WHIT>	Big White Pointers (V4)
<WP_PTR_BLACK>	Small Black Pointers (V4)
<WP_PTR_BLUE>	Blue 3D Pointers (V4)
<WP_PTR_GLOVES>	Glove Pointers (V4)
<WP_PTR_GOLD>	Gold 3D Pointers (V4)
<WP_PTR_GRAY>	Gray 3D Pointers (V4)
<WP_PTR_GREEN>	Green 3D Pointers (V4)
<WP_PTR_LH_BIG_B>	Big Left-Handed Black Pointers (V4)
<WP_PTR_LH_BIG_W>	Big Left-Handed White Pointers (V4)
<WP_PTR_LH_BLACK>	Small Left-Handed Black Pointers (V4)
<WP_PTR_LH_GLOVE>	Left-Handed Glove pointers (V4)
<WP_PTR_LH_WHITE>	Small Left-Handed White Pointers (V4)
<WP_PTR_PINK>	Pink 3D Pointers (V4)
<WP_PTR_RED>	Red 3D Pointers (V4)
<WP_PTR_WHITE>	Small White Pointers (V4)
<WP_PULSE>	Pulse
<WP_RDME>	README (V4)
<WP_RDME>	ReadMe
<WP_READMEFOLDER>	Read Me (V4)
<WP_REFCMDFOLDER>	Reference and Commands

Object ID	Title
	(V4)
<WP_REXREF>	REXX Information
<WP_RJAPPLETPROGREF>	RJAPPLET (V4)
<WP_RS231B>	Remote Support for OS/2 (V4)
<WP_RVRSI>	Reversi (pre V3)
<WP_SCHPAL28>	Scheme Palette (V3)
<WP_SCHPAL96>	Scheme Palette (V4)
<WP_SCHPAL>	Scheme Palette (pre V3)
<WP_SCRBL>	Scramble (pre V3)
<WP_SEEK>	Seek and Scan Files
<WP_SHRED>	Shredder
<WP_SOUND>	Sound
<WP_SPEECH>	VoiceType (V4)
<WP_SPOOL>	Spooler
<WP_SPREAD>	Spreadsheet (pre V3)
<WP_START>	Startup
<WP_STHR>	Start Here (pre V3)
<WP_STICKY>	Sticky Pad (pre V3)
<WP_SYSED>	OS/2 System Editor
<WP_SYSTEM>	System
<WP_TASKSINFO>	Tasks (V4)
<WP_TEMPS>	Templates
<WP_TERM>	PM Terminal (pre V3)
<WP_TODO>	To-Do List (pre V3)
<WP_TOOLS>	Productivity
<WP_TOOLS>	Utilities (V4)
<WP_TOUCH>	Touch
<WP_TRADEMBK>	Trademarks (V3)
<WP_TRBLSHT>	Troubleshooting (V4)
<WP_TROUBLEINFO>	Troubleshooting (V4)
<WP_TUNE>	Tune Editor (pre V3)
<WP_TUTOR>	OS/2 Warp Tutorial (V4)
<WP_UNINST>	Selective Uninstall (V3)
<WP_VIEWER>	Minimized Window Viewer
<WP_VIEWINF>	View (V4)
<WP_WALX>	IBM LAN Distance (V4)
<WP_WALX_SHAD>	IBM LAN Distance (V4)
<WP_WARPCENTER>	WarpCenter (V4)
<WP_WIN2WIN>	WIN-OS/2 Window (V3)
<WP_WINCFG>	*(none - WIN-OS/2 Setup)* (pre V3)
<WP_WINCFG>	WIN-OS/2 Setup (V3)
<WP_WINFS>	WIN-OS/2 Full Screen

Object ID	Title
<WP_WINOS2BK>	Windows Programs in OS/2 (V3)
<WP_addProg>	Guidance on^Adding Program Objects (V4)
<WP_chkIn>	Guidance on^WarpGuide Check-In (V4)
<WP_findObj>	Guidance on^Finding Things (V4)
<WP_selInst>	Guidance on^System Installation (V4)
<WP_sysPrt>	Guidance on^Adding a Printer (V4)
<WP_sysWiz>	Guidance on^System Customization (V4)
<WP_wgSettg>	Guidance on^The WarpGuide (V4)
<WSWIN>	Workspace Manager (V4)
<WS_DHCP_MONITOR>	DHCP Monitor (V4) (V3)

Note: *The object IDs created for printers (<WPPO_...>) will vary depending on the printer queue driver installed.*

See the SysIni() function in the OS/2 *REXX Information* on-line help facility for an example of how to list all of the program objects in your system.

4.2 WPS Classes - Prior to Warp 3

The following hierarchical list contains most, though not necessarily all, of the predefined WPS class names and the name of the DLL module (Dynamic Link Library) which contains the class method, where known. WPS class names do not appear to be case sensitive.

```
Class Name                                        DLL name

WPObject                                              PMWP
   ├── WPAbstract                                     PMWP
   │      ├── Dman                                 DESKMAN
   │      ├── WPClock                             WPCONFIG
   │      ├── WPCountry                           WPCONFIG
   │      ├── WPDisk                                  PMWP
```

```
Class Name                                            DLL name

│   ├── WPKeyboard                                    WPCONFIG
│   ├── WPMouse                                       WPCONFIG
│   ├── WPPalette                                     WPCONFIG
│   │   ├── WPColorPalette                            WPCONFIG
│   │   ├── WPFontPalette                                 PMWP
│   │   └── WPSchemePalette                           WPCONFIG
│   ├── WPPower                                       WPCONFIG
│   ├── WPPrinter                                      WPPRINT
│   │   └── WPRPrinter                                 WPPRINT
│   ├── WPProgram                                         PMWP
│   ├── WPShadow                                          PMWP
│   │   └── WPSharedDir                                   PMWP
│   ├── WPShredder                                        PMWP
│   ├── WPSound                                       WPCONFIG
│   ├── WPSpecialNeeds
│   ├── WPSpool                                        WPPRINT
│   ├── WPSystem                                      WPCONFIG
│   ├── WPTouch                                            TCP
│   └── WPWinConfig                                     WINCFG
├── WPFileSystem                                          PMWP
│   ├── WPDataFile                                        PMWP
│   │   ├── WPBitmap                                      PMWP
│   │   ├── WPIcon                                        PMWP
│   │   ├── WPMet                                         PMWP
│   │   ├── WPPif                                         PMWP
│   │   ├── WPPointer                                     PMWP
│   │   └── WPProgramFile                                 PMWP
│   │       └── WPCommandFile                             PMWP
│   └── WPFolder                                          PMWP
│       ├── ExtendedDeskTop                            EXTDESK
│       ├── WPDesktop                                     PMWP
│       ├── WPDrives                                      PMWP
│       ├── WPFindFolder                                  PMWP
│       ├── WPMinWinViewer                                PMWP
│       ├── WPNetgrp                                      PMWP
│       ├── WPNetwork                                     PMWP
│       ├── WPRootFolder                                  PMWP
│       ├── WPServer                                      PMWP
│       ├── WPStartup                                     PMWP
│       └── WPTemplates                                   PMWP
└── WPTransient                                           PMWP
    ├── PDView                                           WPSPL
    ├── WPDiskCV                                          PMWP
    ├── WPFilter                                          PMWP
    ├── WPFolderCV                                        PMWP
    ├── WPJob
    ├── WPMinWindow                                       PMWP
    ├── WPPort
    ├── WPPrinterDriver                                   PMWP
    └── WPQdr
```

The following list contains other WPS classes that have been identified but whose position in the above structure is unknown.

Mindex	MINXOBJ
MMSound	MMSND
PDView	WPSPL
WPA_mnem	WPNLS
WPCnrView	PMWP
WPIme	WPNLS
WPFdr	WPNLS
WPFinder	PMWP
WPNetLink	PMWP

4.3 WPS Classes - Warp Version 3

The following figure lists the predefined Workplace object classes in a hierarchical order. Each branch in the tree represents an immediate descendant (subclass) of a Workplace object class. The predefined SOM object class, SOMObject, is the root class for all SOM object classes, including all Workplace object classes. The class definition files (.IDL) can be found in the Warp toolkit in ..\WARPTLKT\TOOLKIT\IDL.

```
Class Name                           Class Definition Files

WPObject                                      wpobject.idl
    ├── WPAbstract                               wpabs.idl
    │       ├── Dman                          DESKMAN.DLL
    │       ├── WPClock                       wpclock.idl
    │       ├── WPCountry                      wpctry.idl
    │       ├── WPDisk                         wpdisk.idl
    │       ├── WPLaunchPad                  wplnchpd.idl
    │       ├── WPKeyboard                    wpkeybd.idl
    │       ├── WPMouse                       wpmouse.idl
    │       ├── WPPalette                     wppalet.idl
    │       │       ├── WPColorPalette       wpclrpal.idl
    │       │       ├── WPFontPalette        wpfntpal.idl
    │       │       └── WPSchemePalette      wpscheme.idl
    │       ├── WPPower                       wppower.idl
    │       ├── WPPrinter                     wpprint.idl
    │       ├── WPProgram                       wppgm.idl
    │       ├── WPShadow                     wpshadow.idl
    │       │        └── WPNetLink           wpnetlnk.idl
    │       ├── WPShredder                    wpshred.idl
    │       ├── WPSound                       wpsound.idl
    │       ├── WPSpecialNeeds               wpspneed.idl
    │       ├── WPSpool                       wpspool.idl
    │       └── WPSystem                     wpsystem.idl
    ├── WPFileSystem                           wpfsys.idl
    │       ├── WPDataFile                    wpdataf.idl
    │       │        ├── WPBitmap            wpbitmap.idl
    │       │        ├── WPIcon                wpicon.idl
    │       │        ├── WPMet                  wpmet.idl
    │       │        ├── WPPif                  wppif.idl
    │       │        ├── WPPointer              wpptr.idl
```

```
Class Name                               Class Definition Files

  │     │      └── WPProgramFile               wppgmf.idl
  │     │             └── WPCommandFile        wpcmdf.idl
  │     ├── WPFolder                         wpfolder.idl
  │     │      ├── ExtendedDeskTop            EXTDESK.DLL
  │     │      ├── WPDesktop                   wpdesk.idl
  │     │      ├── WPDrives                  wpdrives.idl
  │     │      ├── WPMinWinViewer                wpmwv.idl
  │     │      ├── WPNetgrp                  wpnetgrp.idl
  │     │      ├── WPNetwork                 wpnetwrk.idl
  │     │      ├── WPRootFolder               wprootf.idl
  │     │      ├── WPServer                  wpserver.idl
  │     │      ├── WPSharedDir                wpshdir.idl
  │     │      ├── WPStartup                  wpstart.idl
  │     │      └── WPTemplates                wptemps.idl
  │     └── WPWinConfig                        wincfg.idl
  └── WPTransient                             wptrans.idl
        ├── WPJob                               wpjob.idl
        ├── WPPort                             wpport.idl
        ├── WPPdr                               wppdr.idl
        └── WPQdr                               wpqdr.idl
```

4.4 WPS Classes - Warp Version 4

The following figure lists the predefined Workplace object classes in a hierarchical order. Each branch in the tree represents an immediate descendant (subclass) of a Workplace object class. The predefined SOM object class, SOMObject, is the root class for all SOM object classes, including all Workplace object classes. The class definition files (.IDL) can be found in the Warp toolkit in ..\TOOLKIT\IDL.

```
Class Name                               Class Definition Files

SOMObject                                       somobj.idl
  ├───SOMClass                                  somcls.idl
  ├───SOMClassMgr                                somcm.idl
  │     └─WPClassManager                      wpclsmgr.idl
  └───WPObject                                wpobject.idl
        ├──────WPAbstract                        wpabs.idl
        │      ├── SmartCenter               [1]SCENTER.DLL
        │      ├── WPClock                      wpclkm.idl
        │      ├── WPCountry                    wpctry.idl
        │      ├── WPDisk                       wpdisk.idl
        │      ├── WPLaunchPad                wplnchpd.idl
        │      ├── WPKeyboard                  wpkeybd.idl
        │      ├── WPMouse                     wpmouse.idl
        │      ├── WPPalette                   wppalet.idl
        │      │      ├── WPColorPalette      wpclrpal.idl
        │      │      ├── WPFontPalette       wpfntpal.idl
        │      │      └── WPSchemePalette     wpscheme.idl
        │      │
```

```
      │     ├── WPPower                 wppower.idl
      │     ├── WPPrinter               wpprint.idl
      │     │     └── WPRPrinter       wprprint.idl
      │     ├── WPProgram                 wppgm.idl
      │     ├── WPShadow               wpshadow.idl
      │     │     └── WPNetLink        wpnetlnk.idl
      │     ├── WPShredder              wpshred.idl
      │     ├── WPSound                 wpsound.idl
      │     ├── WPSpecialNeeds         wpspneed.idl
      │     ├── WPSpool                 wpspool.idl
      │     ├── WPSystem               wpsystem.idl
      │     └── WPWinConfig              wincfg.idl
      ├── WPFileSystem                   wpfsys.idl
      │     ├── WPDataFile              wpdataf.idl
      │     │     ├── WPHtml             wphtml.idl
      │     │     ├── WPIcon             wpicon.idl
      │     │     ├── WPImageFile       wpimage.idl
      │     │     │      └── WPBitmap wpbitmap.idl
      │     │     ├── WPMet               wpmet.idl
      │     │     ├── WPPif               wppif.idl
      │     │     ├── WPPointer           wpptr.idl
      │     │     ├── WPProgramFile      wppgmf.idl
      │     │     │      └── WPCommandFwpcmdf.idl
      │     │     └── WPUrl               wpurl.idl
      │     ├── WPFolder               wpfolder.idl
      │     │     ├── WPDesktop          wpdesk.idl
      │     │     ├── WPDrives         wpdrives.idl
      │     │     ├── WPHost             wphost.idl
      │     │     ├── WPHwManager       wphwmgr.idl
      │     │     ├── WPMinWinViewer      wpmwv.idl
      │     │     ├── WPNetgrp         wpnetgrp.idl
      │     │     ├── WPNetwork        wpnetwrk.idl
      │     │     ├── WPRootFolder      wprootf.idl
      │     │     ├── WPServer         wpserver.idl
      │     │     ├── WPSharedDir       wpshdir.idl
      │     │     ├── WPStartup         wpstart.idl
      │     │     ├── WPTemplates       wptemps.idl
      │     │     └── WPUrlFolder      wpurlfdr.idl
      └── WPTransient                   wptrans.idl
            ├── WPJob                     wpjob.idl
            ├── WPDevice               wpdevice.idl
            │     ├── WPDevAudio        wpaudio.idl
            │     ├── WPDevBus            wpbus.idl
            │     ├── WPDevCDRom        wpcdrom.idl
            │     ├── WPDevCPU            wpcpu.idl
            │     ├── WPDevDiskette    wpdskett.idl
            │     ├── WPDevDisplay     wpdisply.idl
            │     ├── WPDevHarddrive   wphrddrv.idl
            │     ├── WPDevKeyboard    wpkeybdd.idl
            │     ├── WPDevMemory         wpmem.idl
            │     ├── WPDevMouse       wpmoused.idl
            │     ├── WPDevParallel     wpparal.idl
            │     ├── WPDevPeriph      wpperiph.idl
            │     ├── WPDevSerial      wpserial.idl
            │     ├── WPDevTape          wptape.idl
```

```
          │      └── WPDevTimer          wptimer.idl
          ├── WPPort                      wpport.idl
          ├── WPPdr                        wppdr.idl
          └── WPQdr                        wpqdr.idl
```

Note 1: The class definition file name is unknown for the SmartCenter class.

4.5 WPS Objects: Key Values / Pairs

Setup strings used by SysCreateObject() and SysSetObjectData() must contain a string which is composed of a series of "key name=value" pairs that create / change the behavior of the object respectively. Key name is not case sensitive for the default WPS objects. For example, you can specify Title and TITLE interchangeably. However, key name values used by other private setup strings, may be case sensitive.

Key names defined below for the WPObject class apply to subordinate classes as well (e.g. WPFolder and WPProgram), unless overridden.

Also, SOM/WPS enabled applications can define additional, private setup strings. An example is the group of key names beginning with SIO_ belonging to Ray Gwinn's SIO / VSIO drivers and shown below under *DOS Settings*.

The following lists include the "key name=value" pairs that have been identified by the author and others. Because of the lack of developer related documentation covering the WPS, there are probably others which do not appear below. Notes concerning these values refer to results obtained with either OS/2 2.1 or OS/2 Warp Version 3 or 4 in ship-level form.

Furthermore, there is no assurance that all of the values are detailed for each key name nor is there an explanation for all key names which have been found within the OS/2 DLL's where the key name was found to be defined.

The DOS and WINOS2 Settings values are grouped separately in the next section since there are unique characteristics associated with them.

ultiple key names are separated by semicolons and multiple lues for a key name are separated by commas.

```
ample:
   "key1=value1;key2=value2,value3;"
```

› specify a literal comma or a literal semicolon inside one of e fields an escape character (^ - '5E'x) must precede the mma or semicolon. For example:

^,	indicates a literal comma.
^;	indicates a literal semicolon.

›te 01: *Any changes which are made to an open Settings notebook via SysSetObjectData() are not necessarily reflected in that notebook until it is closed and reopened.*

›te 02: *If the same key name is specified more than once within a setup string, it generally appears as though the first key name-value pair is the one which prevails; however, that is not always the case.*

›te 03: *Some of the alphabetic values, of the key name=value pairs, shown below have been found to be case sensitive with uppercase being required; therefore, all alphabetic values should be created in uppercase.*

›te 04: *A new line character, '0A'x, can be used to cause a value such as Title to occupy more than one line. Also, it appears that the occurrence of the escape character, ^, also causes a new line to be created; however, 2nd and subsequent escape characters used for this purpose appear to be ignored.*

some cases, it is not clear whether a setup string key name ⁄alue pair was created for Warp Version 4 or whether it isted prior to Warp Version 4 and was just not known until arp Version 4. Therefore, some of the key name / value irs marked (V4) may, in fact, be valid for Warp Version 3.

:low are the setup strings identified for the following classes:

```
   WPColorPalette                                    (V4)
   WPDesktop                                         (V4)
   WPDisk
   WPFolder
   WPFontPalette                                     (V4)
   WPHost                                            (V4)
   WPHtml                                            (V4)
```

WPKeyboard	(V4)
WPLaunchPad	(V3)
WPObject	
WPPalette	
WPPdr	(V4)
WPPrinter	(V3)
WPProgram	
WPRPrinter	(V4)
WPSchemePalette	(V4)
WPUrl	(V4)

WPColorPalette

Key Name	Value	Description (V4)
AUTOSETUP	HIRES	This sets the number of default colors in the color palette to the 256-color Mixed Color Palette.
	LORES	This sets the number of default colors in the color palette to the 16-color Solid Color Palette.
COLORS	RGB values	These are the initial color values of each cell in the color palette. The values for each cell are separated by commas. (This is equivalent to calling the wpSetupCell method.) The RGB value must be presented as a 6-digit hex value in the format 'RRGGBB'x where RR, GG, and BB are the red, green, and blue, values ranging between '00'x and 'FF'x (0-255 decimal).
XCELLCOUNT	columns	Number of X cells as decimal digits. For Solid Color Palette,

WPColorPalette

Key Name	**Value**	**Description (V4)**
		AUTOSETUP=LORES must be specified, and the default value is 8. For Mixed Color Palette, AUTOSETUP=HIRES must be specified, and the default value is 16.
YCELLCOUNT	rows	Number of Y cells as decimal digits. For Solid Color Palette, AUTOSETUP=LORES must be specified, and the default value is 2. For Mixed Color Palette, AUTOSETUP=HIRES must be specified, and the default value is 16.

WPDesktop

Key Name	**Value**	**Description (V4)**
AUTOLOCKUP	YES | <u>NO</u>	Specifies the status of the auto-lockup feature. If set, the keyboard and mouse will automatically lock up after the specified number of minutes of inactivity.
LOCKUPAUTODIM	<u>YES</u> | NO	Specifies the status of the autodim feature. If set, the screen blanks out and a floating lock icon is displayed 2 minutes after the keyboard and mouse are locked.

Key Name	Value	Description (V4)
LOCKUPBACKGROUND		
	N	Image file name; This name must be the fully-qualified path of the image file. "?:\" is permitted to indicate the boot drive. A value of (none) can be used to indicate the absence of an image file.

	M	Image mode; This mode can be one of the following:
		N = Normal image
		T = Titled image
		S = Scaled image

	S	Scaling factor.

	B	Background type; This can be one of the following:
		I = Image
		C = Color only

	C	Background color; This color can be 3 numbers representing RGB values (red, green, blue).

Example:

```
  "BACKGROUND=?:\OS2\BITMAP\OS2LOGO.BMP,S,3,I"
     or
  "BACKGROUND=(none),,,C,255 222 255"
```

Key Name	Value	Description (V4)
LOCKUPFULLSCREEN	<u>YES</u> | NO	Specifies whether the entire screen is taken up by the lockup background image. The lockup background specified by the LOCKUPBACKGROUND keyname is displayed when the system locks the keyboard and mouse. Otherwise, when the system locks up, a message box is to be displayed prompting you to enter your lockup password.
LOCKUPONSTARTUP	YES | <u>NO</u>	Specifies whether or not the keyboard and mouse are automatically locked when the system is started or restarted.
LOCKUPSCREENSAVERMODE	<u>YES</u> | NO	Specifies whether or not a password is required to unlock the keyboard and mouse. *YES* indicates that the lockup facility acts like a screen saver without a password being required. *NO* indicates that a password is required to unlock the keyboard and mouse
LOCKUPTIMEOUT	<u>3</u>	Specifies the number of minutes of keyboard and mouse inactivity that

WPDesktop

Key Name	**Value**	**Description (V4)**
		will cause the system to automatically lock the keyboard and mouse. Value can range from 1 to 99.

WPDisk

Key Name	**Value**	**Description**
DRIVENUM	n	Logical drive number (1-26).

WPFolder

Key Name	**Value**	**Description**
ALWAYSSORT		(V3)
	<u>NO</u> | YES	Indicates whether contents of the folder should be displayed in sorted order.
(OS/2 2.1) BACKGROUND	file_name	Defines the folder background. *File_name* is the name of a file in the \OS2\BITMAP directory of the boot drive or the full file system name of any other .BMP file.

Note 01: *There is no method for altering the other background characteristics for a folder (e.g. image vs. color; normal, scaled or tiled image; etc.) using either SysCreateObject() or SysSetObjectData().*

Notc 02: *There is no method for altering "Always maintain sort order" using either SysCreateObject() or SysSetObjectData().*

WPFolder Key Name	Value	Description
(Warp) BACKGROUND	N,M,S,B,C	(V3) Set folder background, where:
	N	Image file name; This name must be the fully-qualified path of the image file. "?:\" is permitted to indicate the boot drive. A value of (none) can be used to indicate the absence of an image file.
	M	Image mode; This mode can be one of the following: N = Normal image T = Titled image S = Scaled image
	S	Scaling factor.
	B	Background type; This can be one of the following: I = Image C = Color only
	C	Background color; This color can be 3 numbers representing RGB values (red, green, blue).

Example:

```
  "BACKGROUND=?:\OS2\BITMAP\OS2LOGO.BMP,S,3,I"
     or
  "BACKGROUND=(none),,,C,255 222 255"
```

WPFolder Key Name	Value	Description
DEFAULTSORT	0	Sets the default sort value for the folder according to *number*.
	-2	Name
	-1	Type
	0	Folder's popup first item
	5	Real name
	6	Size
	7	Last write date
	9	Last access date
	11	Creation date
DEFAULTVIEW		(V3)
	ICON	Sets the default open view to the ICON (or CONTENTS) view.
	TREE	Sets the default open view to the TREE view.
	DETAILS	Sets the default open view to the DETAILS view.
	DEFAULT	Restores the default view to its original value by undoing any previously selected setting.
	blank	Sets the default open view to the view of the containing folder.
DETAILSCLASS		(V3)
	classname	Set object class for which the details are displayed in details view. The default object class is WPFileSystem.

WPFolder Key Name	Value	Description
DETAILSFONT	font size & face name	Setup string used to define the font associated with the icon details of the folder. For example: 8.Helv.
DETAILSSHADOWCOLOR	color	(V4) Set color of the text for a shadow object in details view. The color value may be the name of a color or 3 numbers representing RGB values (red, green, blue).
DETAILSTEXTCOLOR	color	(V4) Set the color of the text for a normal object in details view. The color value may be the name of a color or 3 numbers representing RGB values (red, green, blue).
DETAILSTODISPLAY	index [,...]	(V4) Set details to be displayed for a given class. The index is the column index, starting at 0, of the field to display. For example, "DETAILSTODISPLAY=0,2" specifies that only the first and third details data items are to be displayed. The default value is to display all fields.

WPFolder Key Name	Value	Description
DETAILSVIEW	style	Set details view to a specified view style from the following:
	NORMAL	Normal size icons.
	MINI	Small icons.
Note 01:	*There is no equivalent notebook tab setting for DETAILSVIEW which provides the above option.*	
Note 02:	*MINI is an example of one of the key word values found to be case sensitive.*	
ICONFILE	index,file_name	(V3) *File_name* is a full file system name used to set the file name of the animation (closed folder) icon. The "index" value must be set to 1. The specified file contains the folder's closed folder icon.
ICONFONT	font size & face name	Setup string used to define the font associated with the icon view of the folder. For example: 8.Helv.
ICONGRIDSIZE	<u>h,v</u>	(V4) This value sets the horizontal and vertical icon view spacing within a folder. It has a range of 0 to 999 pixels.

WPFolder Key Name	**Value**	**Description**
Note:	*The default values appear to vary by screen resolution.*	
ICONNFILE	index,file_name	(V3) *File_name* is a full file system name used to set the file name of the animation (open folder) icon. The "index" value must be set to 1. The specified file contains the folder's open folder icon.
ICONNRESOURCE	index,id,module	(V3) Set resource of the animation (open folder) icon. The *index* value must be set to 1. The *id* is the identity of an icon resource in the *module* dynamic link library (DLL). The specified resource is the folder's open folder icon.
ICONPOS	x,y	Set object's initial icon position. The *x* and *y* values represent the center of the icon's position in the object's folder in percentage coordinates.
Note:	*Appears to be functional only with SysCreateObject() when creating a new object.*	

WPFolder

Key Name	Value	Description
ICONRESOURCE	id,module	Set object's icon. *ID* is the identity of an icon resource within the *module* dynamic link library (DLL).
Note:	*If both ICONFILE and ICONRESOURCE are specified in the same setup string, ICONFILE prevails.*	
ICONSHADOWCOLOR		(V4)
	color	Set color of the text associated with the shadow icons in icon view. The color may be the name of a color or 3 numbers representing RGB values (red, green, blue).
ICONTEXTBACKGROUNDCOLOR		(V4)
	color	Set color of the background for all text displayed in icon view, tree view, and details view. The color may be the name of a color or 3 numbers representing RGB values (red, green, blue).
ICONTEXTCOLOR		(V4)
	color	Set color of the text associated with normal icons in icon view. The color may be the name of a color or 3 numbers representing RGB values (red, green, blue).
ICONTEXTVISIBLE		(V4)

WPFolder Key Name	Value	Description
	<u>YES</u> | NO	Set icon view text visibility property in icon view.
ICONVIEW	s1[,s2,...sn]	Set icon view to a specified view style from the following:
	FLOWED	Flowed icon view.
	NONFLOWED	Non-flowed icon view.
	NONGRID	Non-gridded icon view.
	NORMAL	Normal size icons.
	MINI	Small icons.
	INVISIBLE	No icons.

Note: *When these values are combined, they are separated with a comma (e.g. FLOWED,MINI).*

ICONVIEWPOS	x1,y1,x2,y2	*X1* and *y1* are the percentage coordinates representing the position of the lower left corner of the window containing the icon. X2 and *y2* represent the percentage values of the width and height respectively of the window containing the icon.

Note: *Appears to be functional only with SysCreateObject() when creating a new object.*

WPFolder Key Name	Value	Description
MENUBAR		(V4)
	<u>YES</u> | NO	Determines whether the menu bar is present in an open view of the folder. The menu bar is the area beneath the title bar and normally contains *Folder, Edit, View, ...*
OPEN	SETTINGS	Open settings view when object is created with SysCreateObject() or modified with SysSetObjectData().
	DEFAULT	Open default view when object is created with SysCreateObject() or modified with SysSetObjectData().
	ICON	Icon view will be opened when object is created with SysCreateObject() or modified with SysSetObjectData().
	TREE	Tree view will be opened when object is created with SysCreateObject() or modified with SysSetObjectData().
	DETAILS	Details view will be opened when object is created with SysCreateObject() or modified with SysSetObjectData().

WPFolder Key Name	**Value**	**Description**
REMOVEFONTS		(V3)
	<u>NO</u> | YES	Indicates whether instance fonts should removed from the folder.
SHOWALLINTREEVIEW		(V4)
	<u>YES</u> | NO	Determines whether all objects are shown in tree view or just other folders are shown.
SORTBYATTR		(V4)
	i1[,12...in]	Define the index list of sort attributes used to sort the details view of the folder. Index list is a comma-delimited list from the following values:
	0	Name
	1	Type
	2	Real name
	3	Size
	4	Last write date
	5	Last access date
	6	Creation date
SORTCLASS		(V3)
	classname	Set class object to sort by. The default class object is WPFileSystem.
TREEFONT	font size & face name	Setup string used to define the font associated with the tree view of the folder. For example: 8.Helv.
TREESHADOWCOLOR		(V4)

WPFolder Key Name	Value	Description
	color	Set color of the text for a shadow object in tree view. The color value may be the name of a color or 3 numbers representing RGB values (red, green, blue).
TREETEXTCOLOR		(V4)
	color	Set color of the text for a normal object in tree view. The color value may be the name of a color or 3 numbers representing RGB values (red, green, blue).
TREETEXTVISIBLE		(V4)
	<u>YES</u> | NO	Set tree view text visibility property in icon view.
TREEVIEW	style	Set tree view to a specified view style from the following:
	LINES	Lines in tree view.
	NOLINES	No lines in tree view.
	NORMAL	Normal size icons in tree view.
	MINI	Small icons in tree view.
	INVISIBLE	No icons in tree view.

Note: *When these values are combined, they are separated with a comma (e.g. LINES,NORMAL).*

WPFolder

Key Name	**Value**	**Description**
WORKAREA	<u>NO</u> | YES	Indicates whether the folder will be a workarea folder.

WPFontPalette

Key Name	**Value**	**Description** **(V4)**
AUTOSETUP	YES	Specifies that the font palette is to be reinitialized with the default set of fonts.
FONTS	fonttype	These are the initial fonts for each cell in the font palette. The values for each cell are separated by commas. (This is equivalent to calling the wpSetupCell method.) The *fonttype* value is presented as the point size followed by a period which is then followed by the face name.

Example:
```
'FONTS=10.Helvetica,' ||,
       '9.WarpSans,'  ||,
       '10.System;'
```

XCELLCOUNT	columns	Number of X cells as decimal digits. If not specified, defaults to 2.
YCELLCOUNT	rows	Number of Y cells as decimal digits. If not specified, defaults to 4.

WPHost

Key Name	Value	Description (V4)
HOSTNAME	hostname	Set the hostname to be accessed using an FTP Host object. This value is designated in the "Hostname" field on the Host page. For example: ftp.cfsrexx.com
USERNAME	username	Set the username to be used when accessing a hostname using an FTP Host object. This value is designated in the "Username" field on the Host page.
PASSWORD	password	Set the password to be used to access the given host with a given username. This value is designated in the "Password" field on the Host page. This value is not required when the object is created. If one is not specified, the user will be prompted to enter a password when the host is accessed. If specified, passwords are stored in an encrypted form when set.
ACCOUNT	account	Set the account value to be used when accessing a given hostname or username using the FTP Host object. This value is designated in the

Key Name	Value	Description (V4)
		"Account" field on the Host page. This value is required only when the FTP server being accessed maintains account information for host accesses.
FILETRANSFERTYPE	ASCII	Set the default file transfer mode for an FTP Host object. This value is designated by selecting the ASCII "Default download type" radio button on Host page 1.
	BINARY	This is the default file transfer mode. This value is designated by selecting the BINARY "Default down-load type" radio button on Host page 1.
REMOTEDIR	path	Specify which directory will be used as the initial working directory when connecting to a host system using the specified FTP Host object. For example: e:\public\bin,., This value is designated in the "Preferred remote directory" field on the Host page 2. The syntax

Key Name	Value	Description (V4
		of this path specification must be in a format understood by the remote host's operating system. The given username and account must have permissions set to access this directory on the remote host.
LOCALDIR	path	Specify the directory to be used as the default download directory for GET operations using the FTP Host object if one is not explicitly indicated. This value is designated in the "Preferred local (download) directory" field on Host page 2.
INCLUDE	pattern	This is used to filter remote files and directories from the FTP Host object's open views. This value is designated in the "Onl display files matching pattern" field on the Include page. The syntax of the pattern must be understood by the remote host's operating system.

WPKeyboard Key Name	Value	Description (V4)
CursorBlinkRate	-1	Sets the rate at which the cursor blinks. This value must be in the range 0 to 890 where 890 is the slowest.
EditTitleTextKey	0	Key combination used to edit title text for an object. The default key pair is <Shift-F9>.
KeyRepeatDelay	-1	Sets the time before the key starts repeating. This value must be in the range 0 to 890.
KeyRepeatRate	0	Sets the rate at which a key repeats. This value must be in the range 1 to 20.
PopUpMenu	0	Key combination used to activate a object's context sensitive menu. The default key pair is <Shift-F10>.

WPLaunchPad

Key Name	**Value**	**Description (V3)**
DRAWEROBJECTS	n,object [, ...]	A comma delimited list, each of which represents a drawer number followed by either object IDs or fully qualified path names. The drawer number (0 = LaunchPad itself, 1 = first drawer, etc.) is separated from the object name with a comma.
FPOBJECTS	<object ID> | file_name [, ...]	A comma delimited list of objects to be added to the end of the LaunchPad. The value(s) for this key name is/are object IDs or fully qualified path names.
Note:	*This is equivalent to specifying DRAWEROBJECTS with a drawer number of 0.*	
LPACTIONSTYLE	<u>TEXT</u>	Display action buttons as text.
	MINI	Display action buttons as mini icons.
	NORMAL	Display action buttons as normal sized icons.
	OFF	Do not display action buttons.

WPLaunchPad

Key Name	Value	Description (V3)
LPCLOSEDRAWER	<u>NO</u> | YES	Specifies whether a drawer should be closed after a LaunchPad object is opened.
LPDRAWERTEXT	<u>NO</u> | YES	Specifies whether object titles should be displayed on the LaunchPad drawers (not buttons).
LPFLOAT	<u>NO</u> | YES	Specifies whether the LaunchPad should float to the top of other windows.
LPHIDECTLS	<u>YES</u> | NO	Specifies whether the LaunchPad frame controls (i.e. title bar and icon) should be hidden.
LPSMALLICONS	<u>NO</u> | YES	Specifies whether small icons or default size icons should be displayed on the LaunchPad.
LPTEXT	<u>NO</u> | YES	Specifies whether object titles should be displayed on the LaunchPad buttons (not drawers).
LPVERTICAL	<u>NO</u> | YES	Specifies whether the LaunchPad should be displayed horizontally or vertically.

Note: *It is not possible to add objects to the LaunchPad under REXX. The LaunchPad must be recreated with any new objects included in the setup string.*

WPObject Key Name	Value	Description
CCVIEW	<u>DEFAULT</u>	The system default value of the concurrent view setting of the system is used when the user selects Open. (V3)
	YES	New or additional (concurrent) views of this object will be created every time the user selects open.
	<u>NO</u>	Open views of this object will resurface when the user selects open.
DEFAULTVIEW		(V3)
	<u>SETTINGS</u>	Set default open view to Settings view.
	id	Set default open view to the *ID* of a user-added view.
HELPLIBRARY	file_name	*File_name* is a full file system name used to set the file name of the help library.
HELPPANEL	id	Set object's default help panel.
HIDEBUTTON	<u>NO</u>	Views of this object will have a minimize button as opposed to a hide button.
	YES	Views of this object will have a hide button as opposed to a minimize button.

WPObject Key Name	Value	Description
Note:	*HIDEBUTTON=YES appears to be equivalent to an older setting of VIEWBUTTON=HIDE. HIDEBUTTON=NO appears to be equivalent to an older setting of VIEWBUTTON=MINIMIZE.*	
ICONFILE	file_name	*File_name* is a full file system name used to define the object's icon.
ICONPOS	x,y	Set object's initial icon position. The *x* and *y* values represent the center of the icon's position in the object's folder in percentage coordinates.
ICONRESOURCE	id,module	Set object's icon. *ID* is the identity of an icon resource within the *module* dynamic link library (DLL).
Note:	*If both ICONFILE and ICONRESOURCE are specified in the same setup string, ICONFILE prevails.*	
Note:	*Appears to be functional only with SysCreateObject() when creating a new object.*	
LOCKEDINPLACE	<u>NO</u> | YES	(V4) If locked in place is set, the object's icon is fixed in position in an open icon view of the folder containing the object.

WPObject Key Name	Value	Description
MENUITEMSELECTED		(V4)
	menu_item	Simulates selecting the specified *menu_item* from the object's popup menu.
MENUS		(V4)
	<u>LONG</u> | SHORT	Set the object's popupmenu to long or short format. Short menus will not include Help and Create Shadow.
MINWIN	DESKTOP	Views of this object will minimize to the Desktop when their minimize button is selected.
	VIEWER	Views of this object will minimize to the minimized window viewer when their minimize button is selected.
	<u>HIDE</u>	Views of this object will hide when their minimize button is selected.
NOCOPY	<u>NO</u> | YES	Resets / sets the object's no copy property.
NODELETE	<u>NO</u> | YES	Resets / sets the object's no delete property.
NODRAG	<u>NO</u> | YES	Resets / sets the object's no drag property.

WPObject Key Name	Value	Description
NODROP	<u>NO</u> | YES	Resets / sets the object's no drop property. When set, no other object can be dropped on it.
Note:	*Appears in PMWP.DLL but does not appear to be functional.*	
NOLINK	<u>NO</u> | YES	Resets / sets the object's no link property.
NOMOVE	<u>NO</u> | YES	Resets / sets the object's no move property.
Note:	*The NOMOVE property does not inhibit an object from being dragged within its current container (i.e the desktop or a folder). It results in a copy rather than a move if the object is dragged to another container.*	
NOPRINT	<u>NO</u> | YES	Resets / sets the object's no print property.
NORENAME	<u>NO</u> | YES	Resets / sets the object's no rename property.
NOSETTINGS	<u>NO</u> | YES	(V3) Resets / sets the object's no settings property, so that the object's settings cannot be opened.

WPObject **Key Name**	**Value**	**Description**
NOSHADOW	<u>NO</u> | YES	Resets / sets the object's no shadow creation property.
NOTVISIBLE	<u>NO</u> | YES	Resets / sets the object's not visible property.
OBJECTID	<name>	Defines the object's identity. The object ID will stay with the object even if it is moved or renamed. An object ID is any unique string preceded with a '<' and terminated with a '>'.

Note 01: *Prior to Warp Version 3, when including the "OBJECTID=<...>" keyname/value pair in a setup string, it must be specified as the last entry in the string.*

Note 02: *Prior to Warp Version 3, an OBJECTID should not be assigned to an object defined as a TEMPLATE since this would lead to multiple objects with the same OBJECTID.*

OPEN	SETTINGS	Open settings view when object is created with SysCreateObject() or modified with SysSetObjectData().
	DEFAULT	Open default view when object is created with SysCreateObject() or modified with SysSetObjectData().

WPObject

Key Name	**Value**	**Description**
SHADOWID	<object ID> | file_name	Specifies the object for which this object is a shadow of. The value for this key name is an object's *object ID* or a fully qualified path name of a directory, program file, or data file.
TEMPLATE	NO | YES	Resets / sets the object's template property.

Note: *Prior to Warp Version 3, an OBJECTID should not be assigned to an object marked as a template since this would lead to multiple objects with the same OBJECTID.*

TITLE	Title	Set object's title.

WPPalette

Key Name	**Value**	**Description (V4)**
XCELLCOUNT	columns	Number of columns of cells. If not specified, defaults to 13.
YCELLCOUNT	rows	Number of rows of cells. If not specified, defaults to 9.
XCELLWIDTH	width	Width in dialog units of each cell.
YCELLHEIGHT	height	Height in dialog units of each cell.

WPPalette

Key Name	**Value**	**Description (V4)**
XCELLGAP	gap	X separation in dialog units between each cell.
YCELLGAP	gap	Y separation in dialog units between each cell.

WPPdr

Key Name	**Value**	**Description (V4)**
INSTPATH	path	Indicates the name of the install directory. When the printer driver object is created, and the driver is not already installed, the required driver will be installed from this *path*.
PORTNAME	port_name	Specifies the port name required by the Windows printer driver. *Port_name* may be one of the following values: COM1, COM2, COM3, COM4, FILE, LPT1, LPT2, LPT3.

Note: *The WINOS2 keyname (page 157) for this object should also be specified when using this keyname.*

PRINTDRIVER	driver_name	Specify the full driver name for the printer driver object. A full driver name is in the driver.device format. For example: IBM42XXX.IBM 4201 Proprinter III

WPPdr

Key Name	**Value**	**Description** **(V4)**
PROMPT	<u>YES</u> | NO	Specifies whether or not the user should be prompted for installation diskettes. When creating a printer driver object, and the required driver is not already installed, the user will be prompted for printer driver installation diskettes.
WINOS2	<u>YES</u> | NO	Specifies whether or not the WINOS2 (or Windows) driver should also be installed when creating the OS/2 printer driver object. The PORTNAME keyname should also be specified when using this keyname.

WPPrinter

Key Name	**Value**	**Description** **(V3)**
APPDEFAULT	<u>YES</u> | NO	This PrintObject is, or is not, to become the application's default PrintObject for printing.
DEFAULTVIEW	DETAILS | ICON	Specifies the default open view for this PrintObject.

WPPrinter

Key Name	Value	Description (V3)
JOBDIALOGBEFOREPRINT	NO | YES	Specifies whether the job properties dialog is displayed before printing.
JOBPROPERTIES	file_name	The complete path to a binary file containing the default job properties for this PrintObject. This file can be created by saving the PRQINFO3->pDriverData data to a file; this data can be obtained by using the SplQueryQueue API of the spooler. For more information about spooler functions, see the Presentation Manager Programming Reference.
OUTPUTTOFILE	NO | YES	Specifies if the output of this PrintObject goes to a file. The user will be prompted for a file_name each time a print job is submitted to this PrintObject.
PORTNAME	portname	The names of already installed ports (i.e LPTx, COMx) to which this PrintObject is to be attached. In the case of more than one port, specify a comma-separated list.

WPPrinter **Key Name**	**Value**	**Description (V3)**
PRINTDRIVER	driver.device	The complete name of the print driver that this PrintObject is to use. For example: 'IBM42XX.IBM 420 Proprinter III', 'LASERJET.HP LaserJet Series II'. In the case of more than one print driver, specify a comma-separated list. These printer drivers must already be installed.
PRINTERSPECIFICFORMAT	YES	The PrintObject spools print jobs in PM_Q_RAW format.
	NO	The PrintObject spools print jobs in PM_Q_STANDARD format.
PRINTWHILESPOOLING	<u>NO</u> | YES	Printing is not, or is, enabled while the job is spooling.
QSTARTTIME	time	The time when the PrintObject starts printing. The time format is HH:MM, and the base is a 24-hour clock.
QSTOPTIME	time	The time when the PrintObject is to stop printing. The time format is HH:MM, and the base is a 24-hour clock.

WPPrinter

Key Name	Value	Description (V3)
QUEUENAME	queue_name	The local *queue_name* for the PrintObject. If a queue name is not specified, one is created by the PrintObject. The *queue_name* key will be ignored if this object has already been assigned a queue.
QUEUEDRIVER	qdrvname	The queue driver name. The queue driver must already be installed and will usually be PMPRINT.
SEPARATORFILE	file_name	A separator file that prints before each print job.
SYNCJOBPROP	<u>YES</u> | NO	(V4) Indicates whether the default properties of the printer object are to be synchronized with the network's printer job properties.
SYNCPRINTERPROP	<u>YES</u> | NO	(V4) Indicates whether the printer properties of the printer object are to be synchronized with the network's printer properties.
TAKEDEFAULTS	<u>YES</u> | NO	(V4) Indicates whether the printer object takes the default values when it is created. When default values are not

WPPrinter

Key Name	Value	Description (V3)
		used, a *Create a Printer* dialogue will be displayed.

WPProgram

Key Name	Value	Description

(Key names which are applicable to the WPProgram class and not the WPProgramFile class are indicated with the character ᵖ.)

ASSOCFILTER	filters	Sets the file_name filter for files associated to this program. Multiple filters are separated by commas.
ASSOCTYPE	type	Sets the type of files associated with this program. Multiple types are separated by commas.
Note:	*By appending two commas (,,) to each of the above parameters, any existing settings are preserved.*	
EXENAME	file_name	Sets the name of the program to *file_name*.
MAXIMIZED ᵖ	YES	Start program maximized.
MINIMIZED ᵖ	YES	Start program minimized. (See the note following NOAUTOCLOSE below.)
NOAUTOCLOSE	YES	Leaves the window open upon program termination.
	NO	Closes the window when the program terminates.
Note:	*Under OS/2 2.11, NOAUTOCLOSE=NO (if specified) overrides MINIMIZED=YES. That is, if the former is explicitly specified, MINIMIZED will be forced to NO. Therefore, if MINIMIZED=YES*	

WPProgram Key Name	Value	Description
	is desired, don't specify NOAUTOCLOSE=NO (which is the default anyway).	
OPEN [P]	SETTINGS	Open settings view when object is created with SysCreateObject() or modified with SysSetObjectData().
	DEFAULT	Open default view when object is created with SysCreateObject() or modified with SysSetObjectData().
Note:	*If SETTINGS is specified, the program object's notebook is opened; however, if DEFAULT is specified, the program object is launched (its icon is cross-hatched) and the program appears in the task list but it does not come to the foreground without either a second call to SysSetObjectData() or manual intervention unless the DeskMan/2 extensions are installed on the system.*	
PARAMETERS	params	Sets the parameter list to *params*. *Params* can be any user-supplied string including substitution values which can be any of the following:
	%	Indicates that no parameters (including a default of the full file system name) are to be passed to the program.
	%*	Insert the full file system name.
	%**P	Insert drive and path information without the last backslash (\).
	%**D	Insert drive with ':' or UNC name.

WPProgram Key Name	Value	Description
	%**N	Insert file name without extension.
	%**F	Insert file name with extension.
	%**E	Insert extension without leading dot. In HPFS, the extension always comes after the last dot.
	[ ]	Results in the user being prompted, via a dialogue box, to enter a parameter string. The dialogue box contains the title "Specify Parameters" and shows the full file system name of the program. (The results are identical whether there are, or are not, any spaces between the brackets.)
	[prompt]	Same as with brackets above along with *prompt* being added to the dialogue box.

Note: *If you try to start a program from the pop-up menu of a folder and the program does not start or displays an error message, you can stop the name of the folder from being sent to the program by placing a percent sign (%) in the parameter field.*

PROGTYPE P	PM	Sets the session type to PM.
	FULLSCREEN	Sets the session type to OS/2 full screen.
	WINDOWABLEVIO	Sets the session type to OS/2 windowed.
	VDM	Sets the session type to DOS full screen.

WPProgram Key Name	Value	Description
	WINDOWEDVDM	Sets the session type to DOS windowed.
	WIN	Sets the session type to WIN-OS/2 full screen in 3.1 standard mode.
	WINDOWEDWIN	Sets the session type to WIN-OS/2 windowed.
	SEPARATEWIN	Sets the session type to WIN-OS/2 window running in a separate VDM.
	PROG_31_STD	Sets the session type to standard WIN-OS/2 full screen.
	PROG_31_ENH	Sets the session type to enhanced WIN-OS/2 full screen.
	PROG_31_ENHSEAMLESSVDM	Sets the session type to enhanced WIN-OS/2 windowed, separate session.
	PROG_31_ENHSEAMLESSCOMMON	Sets the session type to enhanced WIN-OS/2 windowed, common session.

The DOS / WIN-OS/2 Settings are detailed in the next section. Those key names apply to the WPProgram class only.

SET	key name / value	*Key name* is the name of any environment variable and *value* is the string assigned to the environment variable for that session.

Note 01: *The use of the SET key name / value pair results in all default environment variables for the session, except WP_OBJHANDLE= and COMSPEC=, to be cleared and only those two along with explicitly set*

WPProgram

Key Name	Value	Description
	variables will exist for the session.	
Note 02:	*Can be used to extend the PATH= environment variable for a particular session; however, all semicolons contained within the string must be escaped with a carat (^) and the terminating semicolon must be present. For example:*	

```
SET PATH=D:\OS2^;D:\OS2\SYSTEM^;OS2\MDOS^;;
```

Key Name	Value	Description
STARTUPDIR	pathname	Sets the working directory to *pathname*.

WPRPrinter

Key Name	Value	Description (V4)
ICON	file_name	The name of the .ICO file to be used as the icon for this object.
NETID	<network>	The full name of the printer resource as it is known to the network. For example: LS:\\DEPTSERV\DEPTPRNT The NETID key will be ignored and FALSE will be returned if this object has already been assigned a NetId.
REFRESHINTERVAL	seconds	Time interval, in seconds, when the printer object is refreshed.
SHOWJOBS	ALL	All jobs are displayed in the printer object.
	OWN	Only the current user's jobs are displayed in the printer object.

WPRPrinter

Key Name	Value	Description (V4)
TAKEDEFAULTS	<u>YES</u> | NO	(V4) Indicates whether the printer object takes the default values when it is created. When default values are not used, a *Create a Printer* dialogue will be displayed.

WPSchemePalette

Key Name	Value	Description (V4)
AUTOSETUP	<u>YES</u> | NO	This automatically sets the palette values to the original system palette settings otherwise the original system palette settings are ignored.
SCHEMES	Scheme_type	These are the initial schemes for each cell in the scheme palette. The values for each cell are separated by commas. (This is equivalent to calling the wpSetupCell instance method.) The *scheme_type* value is presented as the scheme name followed by a colon, followed by an application name in the INI file.
XCELLCOUNT	columns	Number of X cells as decimal digits. If not specified, defaults to 4.

WPSchemePalette

Key Name	Value	Description (V4)
YCELLCOUNT	rows	Number of Y cells as decimal digits. If not specified, defaults to 7.

Example:

```
  SetupString='SCHEMES=Marble:PM_Marble_Colors,'
              'Southwest:PM_Southwest_Colors,'
              'Khaki:PM_Khaki_Colors;'
```

WPUrl

Key Name	Value	Description (V4)
BROWSER	name or pathname	Specifies the executable that will be invoked to display the web page designated in the "Url" field. This field can be either a pathname or a name of a browser in the PATH. EXPLORE.EXE is the default browser for OS/2 Warp. The Java applet viewer (APPLET.EXE) can be used here to view Java applets with the URL object, but be sure to set the "Integrated browser" check box on the Browser page to NO, because APPLET.EXE does not understand the URL-specific browser options.
DEFAULTBROWSER	name or pathname	This sets the default value for BROWSER for all URL objects. This value will be placed in

WPUrl

Key Name	Value	Description (V4)
		the "Path and file name" field on the Browser page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for BROWSER (page 167).
DEFAULTDISPLAYIMAGES	YES | NO	This sets the default value for DISPLAYIMAGES for all URL objects. This value is used for the "Display images while loading" check box of a URL's properties notebook when the Default push button is pressed or when the URL object is first created. When this keyname is set to YES, the "Display images while loading" check box is checked. Also see the definition for DISPLAYIMAGES (page 173).

Key Name	**Value**	**Description (V4)**
DEFAULTEMAILADDRESS	address	This sets the default value for EMAILADDRESS for all URL objects. This value will be placed in the "Electronic mail address" field on the Server page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for EMAILADDRESS.
DEFAULTENABLEPROXY	<u>NO</u> | YES	This sets the default value for ENABLEPROXY for all URL objects. The "Proxy gateway" check box on the Server page of a URL's properties notebook is checked when the Default push button is pressed and when the URL object is first created. Also see the definition for ENABLEPROXY (page 174).
DEFAULTENABLESOCKS	<u>NO</u> | YES	This sets the default value for ENABLESOCKS for all URL objects. The "Socks server" check box on the Server page of a URL's properties notebook is checked when the Default push button is

Key Name	Value	Description (V4)
		pressed or when the URL object is first created. Also see the definition for ENABLESOCKS (page 174).
DEFAULTINTEGRATEDBROWSER	NO | YES	This sets the default value for INTEGRATEDBROWSER for all URL objects. The "Integrated browser" check box on the Browser page of a URL's properties notebook is checked when the Default push button is pressed or when the URL object is first created. Also see the definition for INTEGRATEDBROWSER (page 174).
DEFAULTLOADGRAPHICS	YES | NO	This sets the default value for LOADGRAPHICS for all URL objects. The "Load graphics" check box on the Browser page of a URL's properties notebook is checked when the Default push button is pressed or when the URL object is first created. Also see the definition for LOADGRAPHICS (page 175).
DEFAULTNEWSSERVER		

Key Name	Value	Description (V4)
	newsserver	This sets the default value for NEWSERVER for all URL objects. This value will be placed in the "Newsserver" field on the Server page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for NEWSERVER (page 176).
DEFAULTPALETTEAWARE	<u>NO</u> | YES	This sets the default value for PALETTEAWARE for all URL objects. The "Palette aware" check box on the Web page of a URL's properties notebook is checked when the Default push button is pressed or when the URL object is first created. Also see the definition for PALETTEAWARE (page 176).
DEFAULTPARAMETERS	parameters	This sets the default value for PARAMETERS for all URL objects. This value will be placed in the "Parameters" field on the Browser page of a URL's properties notebook when the Default push button is

Key Name	Value	Description (V4)
		pressed and when the URL object is first created. Also see the definition for PARAMETERS (page 177).
DEFAULTPRESENTATIONMODE	NO | YES	This sets the default value for PRESENTATIONMODE for all URL objects. The "Presentation mode" check box on the Web page of a URL's properties notebook is checked when the Default push button is pressed or when the URL object is first created. Also see the definition for PRESENTATIONMODE (page 177).
DEFAULTPROXYGATEWAY	proxy	This sets the default value for PROXYGATEWAY for all URL objects. This value will be placed in the "Proxy gatewway" field on the Server page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for PROXYGATEWAY (page 177).
DEFAULTSOCKSSERVER		

Key Name	Value	Description (V4)
	socks	This sets the default value for SOCKSERVER for all URL objects. This value will be placed in the "Socks server" field on the Server page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for SOCKSERVER (page 178).
DEFAULTWORKINGDIR	directory	This sets the default value for WORKINGDIR for all URL objects. This value will be placed in the "Working directory" field on the Server page of a URL's properties notebook when the Default push button is pressed and when the URL object is first created. Also see the definition for WORKINGDIR (page 179).
DISPLAYIMAGES	<u>YES</u> | NO	Specifies that the browser show the images as they are being constructed on the page while being received from the server. This is the default value for the "Display images while loading" check box on the Web page.

WPUrl

Key Name	Value	Description (V4)
EMAILADDRESS	address	This is a required field. It specifies the user's return E-mail address to be used by the browser when responding to other users on mail-to fields of web pages or newsgroup articles. This should be the complete internet E-mail address. For example: dgoran@cfsrexx.com Leaving this field blank will prevent you from responding to mail-to fields and newsgroup articles.
ENABLEPROXY	<u>NO</u> | YES	This sets the "Enable proxy?" check box on the Server page of a URL's properties notebook. If the "Enable proxy?" check box is not checked, the proxy server will not be used.
ENABLESOCKS	<u>NO</u> | YES	This sets the "Socks server" check box on the Server page of a URL's properties notebook. If the "Socks server" check box is not checked, the Socks server will not be used.
INTEGRATEDBROWSER		

Key Name	Value	Description (V4)
	YES	This sets the default value for BROWSER for all URL objects. If the "Integrated browser" check box on the Browser page is checked, this specified that the designated executable has been integrated with the URL object. For example, the IBM WebExplorer browser has been integrated, you would specify YES if you are using EXPLORE.EXE. The Java Applet viewer is not integrated if you are using APPLET.EXE. Other browsers might not be integrated with the URL object. Check the documentation provided with your browser.
	NO	If the "Integrated browser" check box is not checked, you will be unable to use certain browser parameters (for example, Palette Aware, Presentation Mode, and Load Graphics) because a non-integrated browser cannot interpret these command line arguments.
LOADGRAPHICS	<u>YES</u> | NO	Specifies that you can load graphics and images on the web page

Key Name	Value	Description (V4)
		specified by the URL if the "Load graphics" check box on the Web page is checked; otherwise, just text will be loaded which will speed up page downloads.
LOCATOR	url	Specifies the Uniform Resource Locator that uniquely identifies each web page designated in the "Uniform Resource Locator (URL)" field on the Web page. The URL is the address for the page on the web. For example: http://www.cfsrexx.com

Note: *The keywords LOCATOR and URL can be used interchangeably.*

Key Name	Value	Description (V4)
NEWSERVER	news	Specifies the host name or IP address of the server that handles newsgroups for the user's company or account. The first example is a host name and the second example is an IP address: news.company.com or 128.35.89.2
PALETTEAWARE	<u>YES</u> | NO	Specifies the default value for BROWSER for all URL objects. This value will be placed in the "Paletteaware" check box on the Web page. Specifies that the OS/2 palette be used to display pages and images.

Key Name	Value	Description (V4)
PARAMETERS	params	Specifies strings to be included on the command line invocation of the designated executable when it is started. If the browser being used permits optional parameters, declare them using this value. This sets the default value for BROWSER for all URL objects. This value will be placed in the "Parameters" field on the Browser page.
PRESENTATIONMODE	NO | YES	Specifies that the browser use the full-screen (non-windowed) mode. This mode is generally used for presentations. This value is used for the "Presentation mode" check box on the Web page. This is the default value, which specifies that the executable use the OS/2 palette to display its pages and images.
PROXYGATEWAY	proxy	Specifies the URL of the server that handles the interface to the web for the user's company or account. It is the fire wall for insulation of a company from the outside world. A proxy port can also be appended to the end of the proxy string if the proxy server supports it. For example: http://proxy.company.com/or http://128.35.89.2/ A proxy port can also be appended to the end of the proxy string if

Key Name	Value	Description (V4)
		the proxy server supports it. For example: http://proxy.company.com:80/ This value will be placed in the "Proxy gateway" field on the Server page. The slash on the end is required syntax. Contact your system administrator for details about using a proxy gateway server on your system.
SOCKSERVER	socks	Specifies the socks support that will provide access the web through a fire wall, which provides for insulation of a company from the outside world. This value is placed in the "Socks server" field on the Server page. For example: http://socks.company.com/ or http:/socks.company.com:80/ The slash on the end is required syntax.
URL	url	Specifies the Uniform Resource Locator that uniquely identifies each web page designated in the "Uniform Resource Locator (URL)" field on the web page. The URL is the address for a page on the web. For example, an URL might be: http://www.cfsrexx.com

Note: *The keywords URL and LOCATOR can be used interchangeably.*

Key Name	Value	Description (V4)
WORKINGDIR	directory	Specifies OS/2 working directory for the executable, if required, and sets the default value for BROWSER for all URL objects. This value will be placed in the "Working directory" field on the Browser page. A working directory is required if the specified browser requires DLLs or other files from a directory not specified in the LIBPATH or other environment variables.

4.6 DOS / WIN-OS/2 Settings

The DOS and WIN-OS/2 Settings are located within the Session tab of a WPProgram object. They are grouped here separately since there are special considerations used in their "key name=value" format.

These values are specified with SET key name=value. For example:

```
    SET DOS_FILES=45;SET DOS_HIGH=1;
```

When the value is alphabetic, it appears that it is case sensitive. For example:

```
    SET DPMI_DOS_API=disabled
```

will not affect the VDM's value; however,

```
    SET DPMI_DOS_API=DISABLED
```

does result in the VDM's setting being changed.

In almost all instances, where the value is shown as either ON or OFF, the actual value used in the SET statement is **1** for ON and **0** for OFF. For example:

```
    SET COM_HOLD=1; (on, default is off)
```

To specify more than one DOS_DEVICE file name, each must be separated with a comma. For example:

```
'...;SET DOS_DEVICE=C:\OS2\MDOS\ANSI.SYS,  →
                          →  C:\OS2\MDOS\EGA.SYS...;'
```

For those key names which do not show a value here, check the DOS Settings tab in the Program tab of the settings notebook of any DOS VDM.

Where possible, any default value is shown as an underlined value; however, defaults can vary on a system-by-system basis because of different hardware configurations, installation selected options and software dependent considerations. The initial values for any given system can be found in ?:\OS2\INSTALL\DBTAGS.DAT on that system.

Beginning with Warp Version 3.0, an encoded version of the settings for any DOS or WIN-OS2 object can be printed or written to an "encoded" file. If the file output option is selected, any file and path name may be specified; however, the default path shown will be the path of the spool file. This encoded file may then be imported into any other DOS or WIN-OS2 object. With care, it is possible to alter these parameter values with an ASCII editor before loading them then into the object's settings.

It appears that these encoded files are composed of a two line header followed by a repeating group of 4 lines per setting and a blank line. The table below shows the apparent format of these lines:

```
s=DCF
i=title of object

p=parameter key word
t=type of field for value (0 - 5)
v=parameter value       (may be followed by comment)
d=default value         (may be followed by comment)
```

Beginning with Warp Version 4, each of the key name=value pairs may be specified as **DosSetting.** concatenated with the key word=value pair as an environment variable within an OS/2 session (windowed or full screen). Each DOS VDM started with the START command will reflect these settings. For example:

```
SET DosSetting.dos_background_execution=0
SET DosSetting.dos_files=90
```

```
start /dos /win
```

Key names associated only with Windows sessions (WIN-OS/2) are included at the end of this section.

DOS Settings

Key Name	Value	Description
AUDIO_ADAPTER_SHARING		
	<u>None</u>	Indicates that a program in this DOS session does not require an audio adapter.
	Optional	Indicates that a program in this DOS session should use an audio adapter if one is available.
	Required	Indicates that a program in this DOS session must have access to an audio adapter.
Note:	*Case must be as shown.*	
COM_DIRECT_ACCESS		
	<u>0</u> | 1	Allow direct access to the COM ports.
COM_HOLD	<u>0</u> | 1	When set on, provides exclusive access to COM ports for the specified VDM, preventing other processes from using the port and preventing the operating system from releasing the port until the VDM terminates.

Key Name	**Value**	**Description**
COM_RECEIVE_BUFFER_FLUSH		
	<u>NONE</u>	Indicates that, for this DOS session, the operating system is to keep data in the received data buffer.
	RECEIVE DATA INTERRUPT ENABLE	Any data in the received data buffer for this DOS session will be discarded whenever the DOS program enables the received data interrupt.
	SWITCH TO FOREGROUND	Any data in the received data buffer for this DOS session will be discarded whenever the DOS program is brought to the foreground (from a background state).
	ALL	Indicates that communications data be discarded when a DOS program enables the received data interrupt or the program is switched to the foreground. When ALL set, both the "RECEIVE DATA INTERRUPT ENABLED" and the "SWITCH TO FOREGROUND" options are enabled.

DOS Settings Key Name	**Value**	**Description**
COM_SELECT	ALL	
	NONE	
	COMn	When set to *COMn* (where n is a value of 1 to 4), the program will be allowed to select and use only the *COMn* communication port.

Note: *All the above settings, beginning with COM_, are removed when the SIO / VSIO drivers from Ray Gwinn are implemented (noted below in connection with the SIO_ settings).*

DOS Settings Key Name	**Value**	**Description**
DOS_AUTOEXEC	?:\AUTOEXEC.BAT	Used to specify a different batch file other than the default.
DOS_BACKGROUND_EXECUTION	1 | 0	Allows or disallows execution of the program when it is in the background.
DOS_BREAK	0 | 1	Disables or enables Ctrl+Break for the specified VDM.
DOS_DEVICE	file_name	Indicates that *file_name* should be added to the VDM as a device driver. Multiple file names are separated with a comma.

Note 01: *If any device drivers are specified, then all device drivers for that object MUST be specified since previous values are deleted regardless of whether specified via*

Key Name	Value	Description
	SysCreateObject() or SysSetObjectData().	
Note 02:	*The default device driver list is taken from CONFIG.SYS. If a customized list of device drivers is specified for an object, the device drivers specified in CONFIG.SYS are ignored.*	
DOS_FCBS	16	Specifies the maximum number of file control blocks (FCBs) which can be opened by applications running in the VDM. Value can range from 0 to 255.
DOS_FCBS_KEEP	8	Specifies the number of file control blocks FCBs that will be protected against automatic closure. Value can range from 0 to 255.
DOS_FILES	20	Specifies the maximum number of file handles which can be opened in a VDM. Default is replaced by *FILES=* value from CONFIG.SYS. Value can range from 20 to 255.
DOS_HIGH	0 | 1	Determines whether DOS is loaded above the 640KB low memory address space.

DOS Settings

Key Name	**Value**	**Description**
DOS_LASTDRIVE	Z	Specifies the highest available logical drive letter for the specified VDM.
DOS_RMSIZE	640	Specifies the DOS memory size in kilobytes (KB). This is the amount of memory which is available to DOS applications. Value can range from 128 to 640 in increments of 16.
DOS_SHELL	?:\OS2\MDOS\COMMAND.COM ?:\OS2\MDOS /P	May be used to specify the DOS command processor, or to add parameters to affect the command processor. ? represents the boot drive.
Note:	*Default is SHELL= value from CONFIG.SYS.*	
DOS_STARTUP_DRIVE	blank	Specifies the location of the DOS kernel to be loaded into the VDM.
DOS_UMB	0 | 1	Specifies whether DOS owns Upper Memory Blocks (UMBs) and manages the loading of device drivers and TSR programs.

DOS Settings

Key Name	**Value**	**Description**
DOS_VERSION	program_name^,major^,minor^,count	Allows the operating system to report a "fake" DOS version number (major - version, minor - modification level, count - number of times to return this level) in order to support applications which check for a DOS version number.
Note 01:	*The comma between the values is escaped with the caret (^ - '5E'x).*	
Note 02:	*Initially set to the values specified in ?:\OS2\INSTALL\DBTAGS.DAT*	
DPMI_DOS_API	AUTO | ENABLED | DISABLED	Determines whether DOS API translation is enabled for the specified VDM.
DPMI_MEMORY_LIMIT	4	Specifies the maximum amount of protected mode memory (in megabytes) available to DPMI applications running in the VDM. Value can range from 0 to 512.

DOS Settings

Key Name	Value	Description
DPMI_NETWORK_BUFF_SIZE	8	Specifies the size, in kilobytes (KB), of the network translation buffer for DPMI programs in this session. Value can range from 1 to 64.
EMS_FRAME_LOCATION	AUTO | NONE | 8000 | 8400 | 8800 | 8C00 | 9000 C000 | C400 | C800 | CC00 | D000 | D400 | D800 | DC00 |	Allows the location of the LIM EMS region to be explicitly changed.
EMS_HIGH_OS_MAP_REGION	32	Provides the capability of adjusting the size of an additional EMS region in KB. Value can range from 0 to 96 in increments of 16.

Note: *Though the available documentation indicates that the default value for this setting is 32, testing yielded a default value of 0.*

Key Name	Value	Description
EMS_LOW_OS_MAP_REGION	384	Set the size, in KB, of the remappable conventional memory available in a VDM. Value can range from 0 to 576 in increments of 16.

DOS Settings

Key Name	Value	Description
EMS_MEMORY_LIMIT	2048	Set the amount of EMS memory, in KB, available to a VDM. Value can range from 0 to 32768 in increments of 16.
HW_NOSOUND	0 | 1	Allows or disallows sounds started by a DOS program.
HW_ROM_TO_RAM	0 | 1	Enabling causes the operating system to copy read-only memory (ROM) and run the copy in 32-bit random access memory (RAM).
HW_TIMER	0 | 1	Allows an application to have direct access to the timer ports and prevents the operating system from trapping, or intercepting, the timer request and emulating a timer.
IDLE_SECONDS	0	Disables the "IDLE_SENSITIVITY" function for a period of time after useful work has been detected. Value can range from 0 to 60.

Key Name	**Value**	**Description**
IDLE_SENSITIVITY	<u>75</u>	The value is the percentage of the maximum possible polling rate the application can perform. If an application polls at a rate higher than this value, it is considered "idle." Value can range from 1 to 100.
INT_DURING_IO	<u>0</u> | 1	When set on, this creates a second thread for the application to use for interrupt handling when the primary thread is busy with I/O operations.
KBD_ALTHOME_BYPASS	<u>0</u> | 1	When enabled, prevents the Alt+Home key sequence from switching the VDM between full screen and windowed mode.
KBD_BUFFER_EXTEND	<u>1</u> | 0	Increases a VDM's keyboard type-ahead buffer size.
KBD_CTRL_BYPASS	<u>NONE</u> ALT_ESC CTRL_ESC	When enabled, inhibits one of the control key sequences, allowing an application in the VDM to use this sequence for its own purposes.

DOS Settings

Key Name	Value	Description
KBD_RATE_LOCK	0 | 1	Prevents a DOS application in a VDM from changing the system keyboard repeat rate.
MEM_EXCLUDE_REGIONS MEM_INCLUDE_REGIONS	blank	These settings are used to specify address ranges which should be protected / included from use by EMS/XMS and direct access by applications.
MOUSE_EXCLUSIVE_ACCESS	0 | 1	This setting allows VDMs to run applications which maintain their own mouse pointers.
NETWARE_RESOURCES	GLOBAL PRIVATE NONE	Created by the Netware requester's VSHELL.SYS virtual device driver. It determines whether a DOS session uses the virtual Netware shell services (GLOBAL), or must use an explicit load of NETX to get shell services (PRIVATE).

Note: *There apparently is a bug in OS/2 2.1 GA which requires that this field be padded on the right with*

Key Name	Value	Description
		spaces for a total field length of 7 (e.g. LEFT('NONE', 7).
PRINT_SEPARATE_OUTPUT	1 | 0	The default value for this setting is On, which separates the printer output for each DOS program that is running in the same DOS session.
PRINT_TIMEOUT	15	Use this setting to adjust the amount of time, in seconds, that the OS/2 V2.X print subsystem waits before forcing a print job to the printer. Value can range from 0 to 3600.
SESSION_PRIORITY	1	(V3) This setting can be used to change the session's priority from 1 (lowest priority) to 32 (highest priority) for the DOS or WIN-OS2 session.

Each of the following key name-pair values, which begins with SIO_, are considered to be private and are both set and used by the SIO / VSIO drivers developed by Ray Gwinn. Refer to the documentation which accompanies these drivers for an explanation of their function.

```
SIO_Allow_Access_COMn          SIO_Mode_RTS
SIO_Idle_Sensitivity           SIO_Mode_StopBits
SIO_Mode_DataBits              SIO_Mode_XON/XOFF
SIO_Mode_DTR                   SIO_Screen_Sync_Kludge
SIO_Mode_FIFO_Load_Count       SIO_Share_Access_With_OS/2
SIO_Mode_IDSR                  SIO_Virtual_DTR_is_HS
SIO_Mode_OCTS                  SIO_Virtual_RTS_is_HS
SIO_Mode_ODSR                  SIO_Virtualize_16550A
```

Key Name	Value	Description
SIO_Mode_Parity		SIO_Virtualize_COM_Ports
TOUCH_EXCLUSIVE_ACCESS	0 | 1	Set ON to give the windowed DOS program exclusive ownership of the touch display. Only the DOS application will receive touch display data, not PM. Mouse emulation in PM is turned off.
VIDEO_8514A_XGA_IOTRAP	1 | 0	When set OFF, unrestricted access to the 8514/A display adapter hardware is allowed. (Only available for systems with 8514/A or XGA display adapters installed.)
VIDEO_FASTPASTE	0 | 1	Speeds up input from other sources than the keyboard.
VIDEO_MODE_RESTRICTION	none CGA MONO	Extends the 640KB DOS address space by limiting video mode support.

Note: *There is a bug in OS/2 2.1 GA which requires that this field be padded on the right with spaces for a total field length of 15 (e.g* LEFT('CGA', 15 *).*

Key Name	Value	Description
VIDEO_ONDEMAND_MEMORY	1 | 0	Reduces swap space requirements for full-screen VDMs.
VIDEO_RETRACE_EMULATION	1 | 0	Simulates the video retrace status port to provide faster access.

Key Name	Value	Description
VIDEO_ROM_EMULATION	1 | 0	Emulates selected INT 10h ROM Video functions.
VIDEO_SWITCH_NOTIFICATION	0 | 1	Notifies a DOS application of a switch to / from full-screen mode. Default is ON for Win-OS/2 sessions.
VIDEO_WINDOW_REFRESH	0.1	Adjusts the window update frequency for a given VDM (tenths of a second). Value can range from 0.1 to 60 (1 minute).
XMS_HANDLES	32	Specifies the number of XMS extended memory block (EMB) handles. Value can range from 0 to 128.
XMS_MEMORY_LIMIT	2048	Specifies the amount of XMS memory, in KB, for this VDM. Value can range from 0 to 16384 in increments of 4.
XMS_MINIMUM_HMA	0	Specifies the minimum HMA memory request allowed in KB. Value can range from 0 to 63.

Windows sessions only:

WIN_ATM	0 | 1	(V3) Disables or enables Adobe Type Manager font support for Win-OS2 programs.
WIN_CLIPBOARD	1 | 0	When set on, this will allow the session to share clipboard information among OS/2, DOS (window), and Windows programs.

DOS Settings

Key Name	**Value**	**Description**
WIN_DDE	1 | 0	When set on, this will enable sharing of data among other OS/2 and Windows programs.
WIN_RUN_MODE		See PROGTYPE settings under the SESSION tab in the WPProgram settings on page 164.

Note: *Windowed vs. full screen windowed sessions are governed by EXENAME (page 161) of the program object being set to PROGMAN.EXE or not.*

5. WPTools

The WPTools API is unique freeware that was developed by Henk Kelder of the Netherlands and is used within his CHECKINI program. CHECKINI is a utility which can be used to maintain OS/2 INI files. WPTools may be downloaded via anonymous FTP from *ftp.cfsrexx.com/pub/wptool*.zip* or with a World Wide Web browser at *www.cfsrexx.com* on the REXX-related files page.

WPTOOLS.DLL is a Dynamic Link Library that contains two extremely useful and unique functions which provide an interface between REXX programs and Workplace Shell objects. Though WPTOOLS.DLL contains only two functions in addition to its registration entry point, those functions are generally unavailable elsewhere.

REXX must be informed of WPTools' presence and, once registered, each WPTools function is available to all other REXX sessions. WPTools can be registered with:

```
    call RxFuncAdd 'WPToolsLoadFuncs, →
                         → 'WPTOOLS', 'WPToolsLoadFuncs'
    call WPToolsLoadFuncs
```

WPToolsQueryObject has code to support (almost) all object classes for which object setup strings are defined, being:

Class	Setup strings returned
WPObject	CCVIEW, DEFAULTVIEW, HELPPANEL, HIDEBUTTON, MINWIN, NOCOPY, NODELETE, NODRAG, NODROP, NOLINK, NOMOVE, NOPRINT, NORENAME, NOSETTINGS, NOSHADOW, NOTVISIBLE, OBJECTID, TITLE
WPAbstract	TEMPLATE
WPProgram	ASSOCFILTER, ASSOCTYPE, EXENAME, MAXIMIZED, MINIMIZED, NOAUTOCLOSE, PARAMETERS, PROGTYPE, SET, STARTUPDIR
WPShadow	SHADOWID
WPRPrinter	NETID (1)
WPPrint	APPDEFAULT, JOBDIALOGBEFOREPRINT, OUTPUTTOFILE, PORTNAME, PRINTDRIVER, PRINTERSPECIFICFORMAT, PRINTWHILESPOOLING, QSTARTTIME,

	QSTOPTIME, QUEUENAME, QUEUEDRIVER, SEPARATORFILE
WPServer	NETID (2)
WPNetgrp	NETID (2)
WPDisk	DRIVENUM
WPFontPalette	FONTS, XCELLCOUNT, YCELLCOUNT, XCELLWIDTH, XCELLHEIGHT, XCELLGAP, YCELLGAP
WPColorPalette	COLORS, XCELLCOUNT, YCELLCOUNT, XCELLWIDTH, XCELLHEIGHT, XCELLGAP, YCELLGAP
WPFileSystem	MENU (3)
WPProgramFile	ASSOCFILTER, ASSOCTYPE, EXENAME, MAXIMIZED, MINIMIZED, NOAUTOCLOSE, PARAMETERS, PROGTYPE, SET, STARTUPDIR
WPFolder	ALWAYSSORT, BACKGROUND, DETAILSCLASS, DETAILSFONTS, ICONFONT, TREEFONT, ICONNFILE, ICONVIEW, SORTCLASS, TREEVIEW, DETAILSVIEW, WORKAREA
WPLaunchPad	All documented setup strings.

(1) Includes settings for WPPrint.
(2) These settings cannot be used to recreate the object.
(3) MENU doesn't work when applying.

For each object, WPToolsQueryObject() returns not only setup string values for the object itself (when supported), but also for all parent classes. When, for example, one uses WPToolsQueryObject() against the Desktop (class WPDesktop) setup strings will be returned from the classes WPFolder, WPFileSystem and WPObject.

5.1 WPTools Functions

WPToolsFolderContent(folder, stem)

Returns 1 indicating that the identity of all of the abstract objects contained in *folder* have been sequentially assigned to the tails of *stem*; otherwise, returns 0.

Folder is either the object ID or the full file system name of the folder / directory to be interrogated.

On successful completion, *stem.0* will contain the number of abstract objects identified. Each entry returned will be either the object ID or a string beginning with the character # followed by the character string notation of the object's handle. (Handles are the internal identification assigned to every object in the WPS.)

WPToolsQueryObject(object, [class], [title], →
→ [setup], [location])

Returns 1 indicating that the specified properties of *object* were retrieved; otherwise, returns 0.

Class, title, setup, and location are optional variable names that will have the respective properties assigned to them. As such, they should be surrounded with quotes to prevent a NOVALUE condition.

Object may contain a full file system name or a string containing the # character followed by the character representation of the object's handle (see WPToolsFolderContent(), page 197, for a description of handles).

Class, if specified, will contain the Workplace Shell class of the object. *Title*, if specified, will contain the object's title. *Setup*, if specified, will contain the setup string which can be used to recreate the object. The setup string will contain the parameters that could be used with SysCreateObject() or SysSetObjectData() to create / modify the object. *Location*, if specified, will contain the object's location.

WPToolsVersion()

Returns the version and modification level o
WPTOOLS.DLL as two numbers separated by a period

6. RxFTP - REXX File Transfer Protocol

The RxFTP API (RXFTP.DLL) provides the facility for a REXX program to utilize the TCP/IP FTP protocol from within a REXX program. Information on the FTP subcommands is contained in the *IBM Transmission Control Protocol/Internet Protocol Version 4 for OS/2: User's Guide* (Warp Version 4 - \tcpip\help\tcpguifde/hlp).

Information on FTP API calls is contained in the *IBM Transmission Control Protocol/Internet Protocol Version 4 for OS/2: Programmer's Reference* (Warp Version 4 - \tcpip\help\tcpcr.hlp).

Most of the REXX FTP API functions correspond to their like-named FTP subcommands.

Opening and Closing Functions:

FtpDropFuncs()	**FtpLoadFuncs()**	**FtpLogoff()**
FtpSetBinary()	**FtpSetUser()**	**FtpVersion()**

File Action Functions:

FtpAppend()	**FtpDelete()**	**FtpGet()**
FtpPut()	**FtpPutUnique()**	**FtpRename()**

Directory Listing Functions:

FtpDir()	**FtpLs()**

Directory Action Functions:

FtpChDir()	**FtpMkDir()**	**FtpPwd()**
FtpRmDir()		

Remote Server Functions:

FtpPing()	**FtpProxy()**	**FtpQuote()**
FtpSite()	**FtpSys()**	

RxFTP must be loaded and REXX must be informed of its presence. This task is accomplished in a similar fashion to REXXUTIL and, once registered, each function is available to all other REXX sessions. RxFTP can be registered with:

```
call RxFuncAdd 'FtpLoadFuncs', →
                          → 'RXFTP', 'FtpLoadFuncs'
call FtpLoadFuncs
```

It can be removed (unregistered) with:

```
    call FtpDropFuncs
```

6.1 RxFTP Return Values

RxFTP return values are divided between "Set errors" anc "FTP Errors". Set errors result from the **FtpSetBinary()** anc **FtpSetUser()** functions. All other functions return FTP error: in the variable **FTPERRNO** if necessary.

Set Error Values

The Set error codes are returned by functions that pass string to RxFTP that, in turn, are used by subsequent calls to RxFT] functions. The Set error codes returned are 1 if a valid strin is passed to the function and 0 if an invalid string is passed t the function.

FTP Error Values

All of the RxFTP functions which result in communicatio between the local program and a remote host return 0 for successful call and -1 if there is an error associated with th RxFTP function call. If -1 is returned by any RxFTP functio call except for **FtpPing()**, the predefined variable **FTPERRN** will contain one of the following strings:

FTPABORT	Transfer stopped
FTPCOMMAND	Command failed
FTPCONNECT	Unable to connect to server
FTPDATACONN	Error initializing data connection
FTPHOST	Unknown host
FTPLOCALFILE	Error opening local file
FTPLOGIN	Login failed
FTPNOPRIMARY	No primary connection for pro› transfer
FTPNOXLATETTBL	No code page translation tabl‹ was loaded
FTPPROXYTHIRD	Proxy server does not support third party transfers
FTPSERVICE	Unknown service
FTPSOCKET	Unable to obtain socket

Note: *FtpPing() return values are unique and a described along with the function.*

6.2 RxFTP Functions

`FtpAppend( local_file, remote_file[, → → BINARY | ASCII] )`

Returns 0 after successfully initiating a copy of *local_file* to a remote host and adding *local_file* to the end of an existing file of the same name on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO.

As an option, you can specify the transfer to occur in binary mode or text (ASCII) mode. If a you do not specify the transfer mode with this call, the mode specified with a previous **FtpSetBinary()** call is used.

The remote host is specified with a previous **FtpSetUser()** function call. *Local_file* specifies the name of the file on the client. *Remote_file* is the name of the file as it is known on the remote host. Case sensitivity is determined by the remote host.

`FtpChDir( directory_name )`

Returns 0 after successfully changing the working directory on the remote host to *directory_name*; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

`FtpDelete( file_name )`

Returns 0 after successfully deleting *file_name* on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

`FtpDir( pattern, 'stem.' )`

Returns 0 after successfully retrieving the long format of the directory information of the working directory on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call. Similar in function to **FtpLs()** which retrieves a short format of the directory information.

Pattern is an editing pattern composed of file name wildcard characters ? and *.

Note 01: The stem parameter value must include the trailing period and the name must be contained within quotes.

Note 02: Since the position of the values returned for each directory line may vary for Unix-style servers by the host file system, care must be used in parsing the field values. The following is a suggested technique that should be valid on most Unix-style file servers:

```
select
   when WORDS( directory_line ) = 1 then
      do
         /* next level directory name */
      end
   when WORDS( directory_line ) = 2 then
      do
         /* total nnn line */
      end
   otherwise
      do
         parse value directory_line with,
            01 directory_indic,
            02 owner_rwx,
            05 group_rwx,
            08 world_rwx  +3,
               directory_depth,
               owner_id,
               group_id,
               file_size,
               month,
               day,
               time_or_year,
               file_name
         file_name = STRIP( file_name )
         parse value file_name with,
            file_name,
            ' -> ',
            redirected_path_and_file_name
      end
end
```

FtpDropFuncs()

Returns a null string after removing the definitions of all RxFTP functions from the operating system.

`FtpGet( local_file, remote_file[, mode ] )`

Returns 0 after successfully coping *remote_file* on the remote host to the path and name specified in *local_file*; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

Mode is either ASCII or BINARY (or an abbreviation of either) and specifies the mode for the file transfer. If *mode* is omitted, a previous call to **FtpSetBinary()** determines the mode of transfer.

Both *local_file* and *remote_file* can contain path directives which will be taken as relative to the current respective directory. *Local_file* can optionally contain a drive designation.

Note: *It appears that the default transfer mode is binary; however, it is recommended that a mode be specified explicitly.*

`FtpLoadFuncs( parameter )`

Returns a null string after registering all of the functions in RXFTP.DLL. The presence of any value, including a null string, as a parameter will inhibit the copyright notice from being displayed when the functions are registered.

`FtpLogoff()`

Returns 0 after ending all FTP sessions with the host, user ID, and account established by **FtpSetUser()**. Repeated calls to this function will always return 0, even if no session is established.

`FtpLs( pattern, 'stem.' )`

Returns 0 after successfully retrieving the short format of the directory information of the working directory on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call. Similar in function to **FtpDir()** which retrieves a long format of the directory information.

Pattern is an editing pattern composed of file name wildcard characters ? and *.

Note 01: The stem parameter value must include the trailing period and the name must be contained within quotes.

`FtpMkDir( directory )`

Returns 0 after successfully creating *directory* on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

`FtpPing()`

Returns information resulting from sending a ping to the remote host. The remote host is specified with a previous **FtpSetUser()** function call. This function tries to resolve the host name through a name server. If a name server is not present, **FtpPing()** searches the TCPIP\ETC\HOSTS file for a matching host name. Returned information can be:

milliseconds	The number of milliseconds it took for the echo to successfully return.
PINGHOST	Unknown host
PINGPROTO	Unknown protocol ICMP
PINGRECV	Receive failed
PINGREPLY	Host does not reply
PINGSEND	Send failed
PINGSOCKET	Unable to obtain socket

```
FtpProxy( target_host, target_userid, →
        target_password, target_account, →
        source_host, source_userid, →
        source_password, source_account, →
        target_file, source_file[, mode ] )
```

Returns 0 after successfully copying *source_file* on th remote host designated by *source_host*, *source_user_ic source_password*, and optionally *source_account* to th remote host designated by *target_host*, *target_user_ic target_password*, and optionally *target_account*; otherwis returns -1 and assigns a value to the predefined variabl FTPERRNO.

If *source_account* and/or *target_account* are not require by their respective hosts, they must be specified as nu strings.

Both *source_file* and *target_file* can specify different file names and can contain path directives.

Mode is either ASCII or BINARY (or an abbreviation of either) and specifies the mode for the file transfer.

Note: *It appears that the default transfer mode is binary; however, it is recommended that a mode be specified explicitly.*

`FtpPut( local_file, remote_file[, mode ] )`

Returns 0 after successfully copying *local_file* to the remote host as *remote_file*; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

Mode is either ASCII or BINARY (or an abbreviation of either) and specifies the mode for the file transfer. If *mode* is omitted, a previous call to **FtpSetBinary()** determines the mode of transfer.

Both *local_file* and *remote_file* can contain path directives which will be taken as relative to the current respective directory. *Local_file* can optionally contain a drive designation.

Note: *It appears that the default transfer mode is binary; however, it is recommended that a mode be specified explicitly.*

`FtpPutUnique( local_file, remote_file[, mode ] )`

Returns 0 after successfully copying *local_file* to the remote host as *remote_file* so long as *remote_file* does not exist; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

Mode is either ASCII or BINARY (or an abbreviation of either) and specifies the mode for the file transfer. If *mode* is omitted, a previous call to **FtpSetBinary()** determines the mode of transfer.

Both *local_file* and *remote_file* can contain path directives which will be taken as relative to the current respective directory. *Local_file* can optionally contain a drive designation.

Note 01: Use of FtpPutUnique() rather than FtpPut() prevents files from unintentionally being over-written on the remote host.

Note 02: It appears that the default transfer mode is binary; however, it is recommended that a mode be specified explicitly.

`FtpPwd( 'current_directory' )`

Returns 0 after successfully assigning the value of the current working directory on the remote host to *current_directory*; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

Note: The variable passed to the function must be enclosed in quotes.

`FtpQuote( string )`

Returns 0 after successfully sending *string* to the remote host as a quoted string; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call. The remote server must be enabled to allow quoted strings or the function will fail.

`FtpRename( old_name, new_name )`

Returns 0 after successfully renaming *old_name* to *new_name* on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

`FtpRmDir( directory_name )`

Returns 0 after successfully removing *directory_name* on the remote host; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call.

`FtpSetBinary( BINARY | ASCII )`

Returns 1 after successfully indicating the default file transfer mode to the RxFTP API; otherwise, returns 0 if the value passed to the function is neither ASCII nor BINARY or a valid abbreviation of either. A valid abbreviation may be just the first character. This default can be overridden by individual RxFTP functions which provide an ASCII vs. BINARY parameter.

`FtpSetUser( host, user_ID[, account ] )`

Returns 1 after successfully passing *host*, *user_ID*, and *account* to RxFTP; otherwise, returns 0 if any of the parameters contain invalid strings. These values remain in effect until one of the following occurs:

1) A successful call to **FtpLogoff()**
2) A successful call to **FtpDropFuncs()** and, if running under CMD./EXE (a command line session), the session is closed.

Note: *It is advisable to call* FtpLogOff() *when communication with a remote host is complete to prevent unauthorized access to the remote host by other programs.*

`FtpSite( string )`

Returns 0 after successfully sending *string* to the remote host as a site string; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous **FtpSetUser()** function call. The remote server must be enabled to allow site strings or the function will fail.

`FtpSys( 'operating_system' )`

Returns 0 after successfully assigning an identification string associated with the operating system on the remote host to *operating_system*; otherwise, returns -1 and assigns a value to the predefined variable FTPERRNO. The remote host is specified with a previous FtpSetUser() function call.

Note 01: This function can be used to test for a valid connection with a remote host.

Note 02: The variable passed to the function must be enclosed in quotes.

FtpVersion('version')

Returns a null string after assigning the current versio level of the RxFTP API to *version*.

Note: *The variable passed to the function must b enclosed in quotes.*

Appendix A: ASCII and IBM Character Set

000	'00'x	null	043	'2B'x	+	085	'55'x	U
001	'01'x	SOH	044	'2C'x	,	086	'56'x	V
002	'02'x	STX	045	'2D'x	-	087	'57'x	W
003	'03'x	ETX	046	'2E'x	.	088	'58'x	X
004	'04'x	EOT	047	'2F'x	/	089	'59'x	Y
005	'05'x	ENQ				090	'5A'x	Z
006	'06'x	ACK	048	'30'x	0	091	'5B'x	[
007	'07'x	BEL	049	'31'x	1	092	'5C'x	\
008	'08'x	BS	050	'32'x	2	093	'5D'x	]
009	'09'x	HTAB	051	'33'x	3	094	'5E'x	^
010	'0A'x	LF	052	'34'x	4	095	'5F'x	_
011	'0B'x	VTAB	053	'35'x	5			
012	'0C'x	FF	054	'36'x	6	096	'60'x	`
013	'0D'x	CR	055	'37'x	7	097	'61'x	a
014	'0E'x	SO	056	'38'x	8	098	'62'x	b
015	'0F'x	SI	057	'39'x	9	099	'63'x	c
			058	'3A'x	:	100	'64'x	d
016	'10'x	DLE	059	'3B'x	;	101	'65'x	e
017	'11'x	DC1	060	'3C'x	<	102	'66'x	f
018	'12'x	DC2	061	'3D'x	=	103	'67'x	g
019	'13'x	DC3	062	'3E'x	>	104	'68'x	h
020	'14'x	DC4	063	'3F'x	?	105	'69'x	i
021	'15'x	NAK				106	'6A'x	j
022	'16'x	SYN	064	'40'x	@	107	'6B'x	k
023	'17'x	ETB	065	'41'x	A	108	'6C'x	l
024	'18'x	CAN	066	'42'x	B	109	'6D'x	m
025	'19'x	EM	067	'43'x	C	110	'6E'x	n
026	'1A'x	SUB	068	'44'x	D	111	'6F'x	o
027	'1B'x	ESC	069	'45'x	E			
028	'1C'x	FS	070	'46'x	F	112	'70'x	p
029	'1D'x	GS	071	'47'x	G	113	'71'x	q
030	'1E'x	RS	072	'48'x	H	114	'72'x	r
031	'1F'x	US	073	'49'x	I	115	'73'x	s
			074	'4A'x	J	116	'74'x	t
032	'20'x	sp.	075	'4B'x	K	117	'75'x	u
033	'21'x	!	076	'4C'x	L	118	'76'x	v
034	'22'x	"	077	'4D'x	M	119	'77'x	w
035	'23'x	#	078	'4E'x	N	120	'78'x	x
036	'24'x	$	079	'4F'x	O	121	'79'x	y
037	'25'x	%				122	'7A'x	z
038	'26'x	&	080	'50'x	P	123	'7B'x	{
039	'27'x	'	081	'51'x	Q	124	'7C'x	¦
040	'28'x	(	082	'52'x	R	125	'7D'x	}
041	'29'x	)	083	'53'x	S	126	'7E'x	~
042	'2A'x	*	084	'54'x	T	127	'7F'x	

Appendix A: ASCII and IBM Character Set

128	'80'x	Ç
129	'81'x	ü
130	'82'x	é
131	'83'x	â
132	'84'x	ä
133	'85'x	à
134	'86'x	å
135	'87'x	ç
136	'88'x	ê
137	'89'x	ë
138	'8A'x	è
139	'8B'x	ï
140	'8C'x	î
141	'8D'x	ì
142	'8E'x	Ä
143	'8F'x	Å
144	'90'x	É
145	'91'x	æ
146	'92'x	Æ
147	'93'x	ô
148	'94'x	ö
149	'95'x	ò
150	'96'x	û
151	'97'x	ù
152	'98'x	ÿ
153	'99'x	Ö
154	'9A'x	Ü
155	'9B'x	¢
156	'9C'x	£
157	'9D'x	¥
158	'9E'x	₧
159	'9F'x	ƒ
160	'A0'x	á
161	'A1'x	í
162	'A2'x	ó
163	'A3'x	ú
164	'A4'x	ñ
165	'A5'x	Ñ
166	'A6'x	ª
167	'A7'x	º
168	'A8'x	¿
169	'A9'x	⌐
170	'AA'x	¬
171	'AB'x	½
172	'AC'x	¼
173	'AD'x	¡
174	'AE'x	«
175	'AF'x	»
176	'B0'x	░
177	'B1'x	▒
178	'B2'x	▓
179	'B3'x	│
180	'B4'x	┤
181	'B5'x	╡
182	'B6'x	╢
183	'B7'x	╖
184	'B8'x	╕
185	'B9'x	╣
186	'BA'x	║
187	'BB'x	╗
188	'BC'x	╝
189	'BD'x	╜
190	'BE'x	╛
191	'BF'x	┐
192	'C0'x	└
193	'C1'x	┴
194	'C2'x	┬
195	'C3'x	├
196	'C4'x	─
197	'C5'x	┼
198	'C6'x	╞
199	'C7'x	╟
200	'C8'x	╚
201	'C9'x	╔
202	'CA'x	╩
203	'CB'x	╦
204	'CC'x	╠
205	'CD'x	═
206	'CE'x	╬
207	'CF'x	╧
208	'D0'x	╨
209	'D1'x	╤
210	'D2'x	╥
211	'D3'x	╙
212	'D4'x	╘
213	'D5'x	╒
214	'D6'x	╓
215	'D7'x	╫
216	'D8'x	╪
217	'D9'x	┘
218	'DA'x	┌
219	'DB'x	█
220	'DC'x	▄
221	'DD'x	▌
222	'DE'x	▐
223	'DF'x	▀
224	'E0'x	α
225	'E1'x	ß
226	'E2'x	Γ
227	'E3'x	π
228	'E4'x	Σ
229	'E5'x	σ
230	'E6'x	µ
231	'E7'x	τ
232	'E8'x	Φ
233	'E9'x	Θ
234	'EA'x	Ω
235	'EB'x	δ
236	'EC'x	∞
237	'ED'x	φ
238	'EE'x	ε
239	'EF'x	∩
240	'F0'x	≡
241	'F1'x	±
242	'F2'x	≥
243	'F3'x	≤
244	'F4'x	⌠
245	'F5'x	⌡
246	'F6'x	÷
247	'F7'x	≈
248	'F8'x	°
249	'F9'x	∙
250	'FA'x	·
251	'FB'x	√
252	'FC'x	ⁿ
253	'FD'x	²
254	'FE'x	■
255	'FF'x	

Appendix B: Codes Returned by SysGetKey

	Only	Shft	Ctrl	Alt	Alt Ctrl	Ctrl Shft	Alt Shft
Backspace	08	08	7F	000E	000E	7F	000E
Cur Down	E050	E050	E091	00A0	00A0	E091	00A0
Cur Down(KP)	0050	32	0091	02	02	0091	02
Cur Left	E04B	E04B	E073	009B	009B	E073	009B
Cur Left (KP)	004B	34	0073	04	04	0073	04
5 (KP)	004C	35	008F	05	05	008F	05
Cur Right	E04D	E04D	E074	009D	009D	E074	009D
Cur Right (KP)	004D	36	0074	06	06	0074	06
Cur Up	E048	E048	E08D	0098	0098	E08D	0098
Cur Up (KP)	0048	38	008D	08	08	008D	08
Delete	E053	E053	E093	00A3	Oops	E093	00A3
Delete (KP)	0053	2E	0093	--	Oops	0093	--
End	E04F	E04F	E075	009F	009F	E075	009F
End (KP)	004F	31	0075	01	01	0075	01
Enter	0D	0D	0A	001C	001C	0A	001C
Enter (KP)	0D	0D	0A	00A6	--	0A	00A6
Escape	1B	1B	--	--	--	--	--
Home	E047	E047	E077	--	0097	E077	0097
Home (KP)	0047	37	0077	07	07	0077	07
Insert	E052	E052	E092	00A2	00A2	E092	00A2
Insert (KP)	0052	30	0092	--	--	0092	--
Page Down	E051	E051	E076	00A1	00A1	E076	00A1
Page Down (KP)	0051	33	0076	03	03	0076	03
Page Up	E049	E049	E084	0099	0099	E084	0099
Page Up (KP)	0049	39	0084	09	09	0084	--
Space Bar	20	20	20	20	20	20	20
Tab	09	000F	0094	00A5	00A5	0094	00A5
F1	003B	0054	005E	0068	0068	005E	0068
F2	003C	0055	005F	0069	0069	005F	0069
F3	003D	0056	0060	006A	006A	0060	006A
F4	003E	0057	0061	006B	006B	0061	006B
F5	003F	0058	0062	006C	006C	0062	006C
F6	0040	0059	0063	006D	006D	0063	006D
F7	0041	005A	0064	006E	006E	0064	006E
F8	0042	005B	0065	006F	006F	0065	006F
F9	0043	005C	0066	0070	0070	0066	0070
F10	0044	005D	0067	0071	0071	0067	0071
F11	0085	0087	0089	008B	008B	0089	008B
F12	0086	0088	008A	008C	008C	008A	008C

Appendix B: Codes Returned by SysGetKey

	Only	Shft	Ctrl	Alt	Alt Ctrl	Ctrl Shft	Alt Shft
0	30	29	--	0081	0081	--	0081
0 (KP - NL)	30	0052	0092	--	--	0092	--
1	31	21	--	0078	0078	--	0078
1 (KP - NL)	31	004F	0075	01	01	0075	01
2	32	40	0003	0079	0079	0003	0079
2 (KP - NL)	32	0050	0091	02	02	0091	02
3	33	23	--	007A	007A	--	007A
3 (KP - NL)	33	0051	0076	03	03	0076	03
4	34	24	--	007B	007B	--	007B
4 (KP - NL)	34	004B	0073	04	04	0073	04
5	35	25	--	007C	007C	--	007C
5 (KP - NL)	35	004C	008F	05	05	008F	05
6	36	5E	1E	007D	007D	1E	007D
6 (KP - NL)	36	004D	0074	06	06	0074	06
7	37	26	--	007E	007E	--	007E
7 (KP - NL)	37	0047	0077	07	07	0077	07
8	38	2A	--	007F	007F	--	007F
8 (KP - NL)	38	0048	008D	08	08	008D	05
9	39	28	--	0080	0080	--	0080
9 (KP - NL)	39	0049	0084	09	09	0084	09
A	61	41	01	001E	001E	01	001E
B	62	42	02	0030	0030	02	0030
C	63	43	- 03	002E	002E	- 03	002E
D	64	44	04	0020	0020	04	0020
E	65	45	05	0012	0012	05	0012
F	66	46	06	0021	0021	06	0021
G	67	47	07	0022	0022	07	0022
H	68	48	08	0023	0023	08	0023
I	69	49	09	0017	0017	09	0017
J	6A	4A	0A	0024	0024	0A	0024
K	6B	4B	0B	0025	0025	0B	0025
L	6C	4C	0C	0026	0026	0C	0026
M	6D	4D	0D	0032	0032	0D	0032
N	6E	4E	0E	0031	0031	0E	0031
O	6F	4F	0F	0018	0018	0F	0018
P	70	50	- 10	0019	0019	- 10	0019
Q	71	51	11	0010	0010	11	0010
R	72	52	12	0013	0013	12	0013
S	73	53	- 13	001F	001F	- 13	001F
T	74	54	14	0014	0014	14	0014
U	75	55	15	0016	0016	15	0016
V	76	56	16	002F	002F	16	002F
W	77	57	17	0011	0011	17	0011
X	78	58	18	002D	002D	18	002D
Y	79	59	19	0015	0015	19	0015
Z	7A	5A	1A	002C	002C	1A	002C

Appendix B: Codes Returned by SysGetKey

					Alt	Ctrl	Alt
	Only	Shft	Ctrl	Alt	Ctrl	Shft	Shft
'	27	22	--	0028	0028	--	0028
* (KP)	2A	2A	0096	0037	0037	0096	0037
+ (KP)	2B	2B	0090	004E	--	0090	004E
,	2C	3C	--	0033	0033	--	0033
-	2D	5F	1F	0082	0082	1F	0082
- (KP)	2D	2D	008E	004A	004A	008E	004A
.	2E	3E	--	0034	0034	--	0034
/	2F	3F	--	0035	0035	--	0035
/ (KP)	2F	2F	0095	00A4	00A4	0095	00A4
;	3B	3A	--	0027	0027	--	0027
=	3D	2B	--	0083	0083	--	0083
= (KP)	3D	3D	--	0083	0083	--	0083
[	5B	7B	1B	001A	001A	1B	001A
\	5C	7C	1C	002B	002B	1C	002B
]	5D	7D	1D	001B	001B	1D	001B
`	60	7E	--	0029	0029	--	0029

The 2 or 4 byte hexadecimal values shown above are returned by SysGetKey() when the respective key, shown in the first column, by itself or along with the combinations of the Shift key, the Ctrl key or the Alt key are depressed. The leading "E0" is returned when the instruction keys (e.g. Home, Page Up, etc.) separate from the numeric keypad are used.

These same instruction keys, which are part of the numeric keypad (noted with KP above and typically white rather than grey), return a 2 byte hexadecimal value with the "E0" replaced with "00". KP NL indicates the keypad number keys with numeric lock set to on.

⌐ indicates the values for the respective keys; however, these values are retained by OS/2 for its own use and not returned to a Rexx program.

Appendix C: REXXLIB Video Attributes

Foreground Colors

0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	brown
7	white
8	gray
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	yellow
15	high intensity white

Background Colors

0	black
16	blue
32	green
48	cyan
64	red
80	magenta
96	brown
112	white

Attribute values are computed by adding the number that represents the foreground color to the number that represents the background color.

If the blinking attribute is enabled, as it is by default, adding 128 will produce blinking text. If the blinking attribute is disabled with the SCRBLINK function, adding 128 will produce text with bright background colors.

Appendix D: REXX Error Codes
(from REX.MSG and REXH.MSG)

Error codes 1 through 114 are reserved for the REXX interpreter. Error codes 115 through 125 are used by REXX subcommands.

Return codes from the REXX macrospace functions are listed at the end of this Appendix.

01 **File Table full**
There are currently too many files open for this session.

Close files which are open but no longer in use.

02 *not used*

03 **Program is unreadable**
03 **Failure during initialization** (OBJ)
An attempt was made to access a program which was either non-existent or locked by another process. Verify the program file's existence and make sure no other process has it locked.

04 **Program interrupted**
The system interrupted execution of a program because of some error, or by user request.

Trap interrupts via CALL ON HALT or SIGNAL ON HALT.

05 **Machine resources exhausted**
05 **System resources exhausted** (OBJ)
While attempting to execute a program, the language processor was unable to obtain the resources it needed to continue execution.

06 **Unmatched "/*" or quote**
A comment or literal string was started but never finished. This may be detected at the end of the program (or the end of data in an INTERPRET instruction) for comments, or at the end of a line for strings.

07 **WHEN or OTHERWISE expected**
Within a SELECT construct, at least one WHEN construct (and possibly an OTHERWISE clause) is expected.

Look for any instruction other than WHEN (or no WHEN construct before the OTHERWISE) in the SELECT construct.

08 **Unexpected THEN or ELSE**
A THEN or an ELSE has been found that does not match a corresponding IF (or WHEN) clause.

Look for a missing END or DO...END in the THEN part of a complex IF...THEN...ELSE construction.

09 **Unexpected WHEN or OTHERWISE**
A WHEN or an OTHERWISE has been found outside of a SELECT construct. It may have been enclosed unintentionally in a DO...END construct by leaving off an END instruction, or an attempt may have been made to branch to it with a SIGNAL instruction.

10 **Unexpected or unmatched END**
There are more END's in the program than DO's and SELECT's, or the END's are wrongly placed so they do not match the DO's and SELECT's. This error will also be generated if an END immediately follows a THEN or an ELSE.

Putting the name of the control variable on END's that close repetitive loops can also help locate this kind of error. A common mistake that causes this error is attempting to jump into the middle of a loop using the SIGNAL instruction. Since the previous DO will not have been executed, the END is unexpected. Also, since SIGNAL deactivates any current loops, it may not be used to jump from one place inside a loop to another.

11 Control stack full

An interpreter limit of levels of nesting of control structures (DO...END, IF...THEN...ELSE, etc.) has been exceeded. This could be due to a looping INTERPRET instruction, which could loop forever. Similarly, a recursive subroutine or internal function that does not terminate correctly could loop forever.

12 Clause too long

The length of the internal or external representation of a clause has exceeded the interpreter's limit.

13 Invalid character in program

The program includes a character outside of a literal (quoted) string that is not a blank or one of the valid alphanumeric/special characters.

14 Incomplete DO/SELECT/IF

On reaching the end of the program (or end of the string in an INTERPRET instruction), it has been detected that there is a DO or SELECT without a matching END, or an IF that is not followed by a THEN clause to execute.

Putting the name of the control variable on END's that close repetitive loops can also help locate this kind of error.

15 Invalid hexadecimal or binary string

Hexadecimal strings may not have leading or trailing blanks, and may only have embedded blanks at byte boundaries. Only the digits 0-9 and the letters a-f and A-F are allowed. Similarly, binary strings may only have blanks added at the boundaries of groups of four binary digits, and only the digits 0 and 1 are allowed.

This error may also be caused by following a literal string by the one-character symbol "X" (for example the name of the variable X) when the string is not intended to be taken as a hexadecimal specification, or by the symbol "B" when the string is not intended to be taken as a binary specification.

Use the explicit concatenation operator, "||", in situations where the "X" or "B" is intended to represent a variable.

Label not found
A SIGNAL instruction has been executed (or an event for which a trap was set has occurred), and the label specified cannot be found in the program.

Unexpected PROCEDURE
A PROCEDURE instruction was encountered which was not the first instruction executed after a CALL or function invocation.

Check for the possibility of "dropping through" into an internal routine rather than invoking it properly.

THEN expected
All IF clauses and WHEN clauses in REXX must be followed by a THEN clause. Some other clause was found when a THEN was expected.

String or symbol expected
Following either the keyword CALL or the sequence SIGNAL ON or SIGNAL OFF, a literal string or a symbol was expected but neither was found.

Symbol expected
In the clauses CALL ON, END, ITERATE, LEAVE, NUMERIC, PARSE, PROCEDURE, and SIGNAL ON, a symbol can be expected. Either it was not present when required, or some other token was found. Alternatively, DROP, and the EXPOSE option of PROCEDURE, expect a list of symbols. Some other token was found.

Invalid data on end of clause
A clause such as SELECT or NOP is followed by some token other than a comment.

22 Invalid character string
This error results if a literal string contains character codes that are not valid in the interpreter. This might be because some characters are "impossible", or because the character set is extended in some way and certain character combinations are not allowed.

23 Invalid data string
This error results if a data string (result of an expression, etc.) contains character codes that are not valid in the interpreter. This might be because some characters are "impossible", or because the character set is extended in some way and certain character combinations are not allowed.

24 Invalid TRACE request
The setting specified on a TRACE instruction starts with a character that does not match one of the valid TRACE settings (i.e., A, C, E, F, I, L, N, O, or R).

25 Invalid sub-keyword found
An unexpected token has been found in the position in an instruction where a particular sub-keyword was expected. For example, in a NUMERIC instruction, the second token must be DIGITS, FUZZ, or FORM, and anything else is an error.

26 Invalid whole number
One of the following did not evaluate to a whole number:

- the positional patterns in parsing templates
- the power value (right-hand operand) of the power operator
- the values in a DO instruction after the FOR modifier - the values given for DIGITS or FUZZ in the NUMERIC instruction
- the number used in the TRACE setting

This error is also raised if the value is not permitted (for example, a negative repetition count in a DO instruction), or when the division performed during an integer divide or remainder operation does not result in a whole number.

27 **Invalid DO syntax**
Some syntax error has been found in the DO instructio
This might be by using BY, TO, or FOR twice, or usir
BY, TO, or FOR when there is no control variab
specified, etc.

28 **Invalid LEAVE or ITERATE**
A LEAVE or ITERATE instruction was encountered i
an invalid position. Either no loop is active, or the nam
specified on the instruction does not match the contr
variable of any active loop. Note that since intern
routines and the INTERPRET instruction protect D
loops, they become inactive, and therefore a LEAVE i
a subroutine cannot affect a DO loop in the callin
routine. A common cause for this error message
attempting to use the SIGNAL instruction to transf
control within or into a loop. Since SIGNAL terminat
all active loops, an ITERATE or LEAVE would then b
in error.

29 **Environment name too long**
The environment name specified by the ADDRES
instruction is longer than permitted for the system und
which the interpreter is executing.

30 **Name or string too long**
A variable name or a label name (or the length of
literal string) has exceeded the interpreter's limit.

31 **Name starts with number or "."**
A value may not be assigned to a variable whose nam
starts with a numeric digit or a period (since if it wer
permitted one could re-define numeric constants).

32 ***not used***

33 **Invalid expression result**
The result of an expression in an instruction was foun
to be invalid in the particular context in which it wa
used.

Check for an illegal FUZZ or DIGITS value in
NUMERIC instruction (FUZZ may not become large
than DIGITS).

34 **Logical value not 0 or 1**
The expression in an IF, WHEN, DO WHILE, or DO UNTIL phrase must result in a '0' or a '1', as must any term operated on by a logical operator.

35 **Invalid expression**
This is due to a grammatical error in an expression, such as ending it with an operator, or having two operators adjacent with nothing in between. It may also be due to an expression that is missing when one is required.

Check for special characters (such as operators) in an intended character expression which are not enclosed in quotes.

36 **Unmatched "(" or "[" in expression**
This is due to not pairing parentheses or brackets correctly within an expression. There are more left parentheses or brackets than right parentheses or brackets.

37 **Unexpected ",", ")", or "]"**
Either a comma has been found outside a function invocation, or there are too many right parentheses or brackets in an expression.

38 **Invalid template or pattern**
Within a parsing template, a special character that is not allowed (for example, "%") has been found, or the syntax of a variable pattern is incorrect (i.e., no symbol was found after a left parenthesis). This error may also be raised if the WITH sub-keyword is omitted in a PARSE VALUE instruction.

39 **Evaluation stack overflow**
The expression is too complex to be evaluated by the language processor.

Check for too many nested parentheses, functions, etc.

40 **Incorrect call to routine**
The specified built-in or external routine does exist, but it has been used incorrectly. Either invalid arguments were passed to the routine, or the program invoked was not compatible with the language processor, or more than an implementation-limited number of arguments were passed to the routine.

41 **Bad arithmetic conversion**
One of the terms involved in an arithmetic operation is not a valid number, or its exponent exceeds the implementation limit (often 9 digits).

42 **Arithmetic overflow/underflow**
The result of an arithmetic operation requires an exponent that is outside the range supported by the interpreter. This can happen during evaluation of an expression (commonly an attempt to divide a number by 0), or possibly during the stepping of a DO loop control variable.

43 **Routine not found**
A function has been invoked within an expression (or a subroutine has been invoked by a CALL) but it cannot be found. No label with the specified name exists in the program, it is not the name of a built-in function, and the language processor has been unable to locate it externally.

Check for:

- a mis-typed label or name
- a symbol or literal string adjacent to a '(' when it should have been separated by a blank or some other operator (this would be understood as a function invocation).

44 **Function or message did not return data**
An external function has been invoked within an expression, but even though it appeared to end without error, it did not return data for use within the expression.

45 **No data specified on function RETURN**
The program has been called as a function, but an attempt is being made (by RETURN;) to return without passing back any data. Similarly, if an internal routine is called as a function then the RETURN instruction that ends it must specify an expression.

46 **Invalid variable reference**
Within a DROP, PARSE or PROCEDURE instruction, the syntax of a variable reference is incorrect. This may be due to a missing parenthesis or an incorrectly coded variable within the parentheses.

47 **Unexpected label**
A label appeared as part of the instructions executed by an INTERPRET instruction.

Remove the label from the interpreted data.

48 **Failure in system service**
A system service used by the language processor (such as stream input or output, or manipulation of an external data queue) has failed to work correctly and hence normal execution cannot continue.

49 **Interpretation error**
Some kind of severe error has been detected within the language processor or execution process during internal self-consistency checks.

90 **External name not found** (OBJ)
An external class, method, or routine (specified with the EXTERNAL option on a ::CLASS, ::METHOD, or ::ROUTINE directive cannot be found.

91 **No result object** (OBJ)
A message term requires a result object, but the method did not return one.

93 **Incorrect call to method** (OBJ)
The specified method or built-in or external routine does exist, but you used it incorrectly. The associated error subcode will give the specific reason for the error.

97 **Object method not found** **(OBJ)**
This message indicates that the object does not have ε method with the given name. A frequent cause of thi: error is an uninitialized variable.

98 **Execution error** **(OBJ)**
This message indicates that the language processoı detected one of certain errors in execution. The associated error will give the specific reason for the error.

99 **Translation error** **(OBJ)**
This message indicates that some error was detected iı the language syntax. The associated error subcode identifies the specific syntax error.

Note: *Message numbers 50 - 114 are unused in Classic REXX. Only the messages shown from 90 - 9 are used in this range in Object REXX.*

115 RXSUBCOM accepts the following parameters:
To Register a subcommand environment:
```
RXSUBCOM REGISTER ENVIRONMENT_NAME →
                          → DLL_NAME ENTRY_POINT
```

To Query a specific subcommand environment for existence:
```
RXSUBCOM QUERY [ENVIRONMENT_NAME [DLL_NAME]]
```

To Drop a subcommand environment handler:
```
RXSUBCOM DROP ENVIRONMENT_NAME [DLL_NAME]
```

To Load a subcommand environment from disk:
```
RXSUBCOM LOAD ENVIRONMENT_NAME [DLL_NAME]
```

Note: *Check the RXSUBCOM parameters and retry.*

116 The RXSUBCOM parameter REGISTER is incorrect.
RXSUBCOM REGISTER requires all of the following parameters:
```
REGISTER ENVIRONMENT_NAME DLL_NAME ENTRY_POINT
```

Environment_name is the name of the subcommand environment.

DLL_name is the Dynamic Link Library module name.

Entry_point is the name of the function to be executed when called.

Note: *Check the RXSUBCOM parameters and retry the command.*

117 The RXSUBCOM parameter DROP is incorrect.
RXSUBCOM DROP requires the environment name be specified:
```
RXSUBCOM DROP ENVIRONMENT_NAME [DLL_NAME]
```

Environment_name is the name of the subcommand environment.

DLL_name is the Dynamic Link Library module name (optional).

Note: *Check the RXSUBCOM parameters and retry the command.*

118 The RXSUBCOM parameter LOAD is incorrect.
RXSUBCOM LOAD requires the environment name be specified.

```
RXSUBCOM LOAD ENVIRONMENT_NAME [DLL_NAME]
```

Environment_name is the name of the subcommand environment.

DLL_name is the Dynamic Link Library module name (optional).

Note: *Check the RXSUBCOM parameters and retry the command.*

119 The REXX queuing system is not initialized.
The queuing system requires a housekeeping program to run. This program usually runs under the Presentation Manager shell. The program is not running.

Note: *Report this message to your IBM service representative or facility.*

120 The size of the data is incorrect.
The data supplied to the RXQUEUE command is too long. The RXQUEUE.EXE program accepts data records containing 0 - 65472 bytes. A record exceeded the allowable limits.

Note: *Use shorter data records.*

121 Storage for data queues is exhausted.
The queuing system is out of memory. No more storage is available to store queued data.

Note: *Delete some queues or remove queued data from the system. Then retry your request.*

122 The name %1 is not a valid queue name.
The queue name contains an invalid character. Only the following characters may appear in queue names:

```
'A' .. 'Z', '0' .. '9', '.', '!', '?', '_'
```

Note: *Change the queue name and retry the command.*

123 The queue access mode is not correct.
An internal error occurred in RXQUEUE. RXQUEUE.EXE tried to access a queue with an incorrect access mode. Correct access modes are LIFO and FIFO.

Note: *Report this message to your IBM service representative or facility.*

124 The queue %1 does not exist.
The command attempted to access a nonexistent queue. Create the queue and try again, or use a queue that has been created.

125 The RXSUBCOM parameter QUERY is incorrect.
RXSUBCOM QUERY requires the environment name be specified.

```
RXSUBCOM QUERY ENVIRONMENT_NAME [DLL_NAME]
```

Environment_name is the name of the subcommand environment.

DLL_name is the Dynamic Link Library module name (optional).

Note: *Check the RXSUBCOM parameters and retry the command.*

Macrospace return codes

0	The call to the function completed successfully.
1	There was not enough memory to complete th requested function.
2	The requested function was not found in th macrospace.
3	An extension is required for the macrospace file name.
4	Duplicate functions cannot be loaded from a macrospac file.
5	An error occurred accessing a macrospace file.
6	A macrospace save file does not contain valid functio images.
7	The requested file was not found.
8	An invalid search order position request flag was used.

Reader Notes

Tear here

Order Form

Orders only • 1-800-REXXOS2
1-800-739-9672
On-line • www.cfsrexx.com

C F S Nevada, Inc.
953 E. Sahara Avenue, Suite 9B
Las Vegas, Nevada 89104-3012
702-732-9616 Voice
702-732-3847 FAX

Name: ____________________

Company: ____________________

Address: ____________________

City, State and Zip: ____________________

Voice / FAX number: ____________ Email Address: ____________

AMEX Discover MC Visa Card # _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Expires: ____/____ Check # ________

Quantity: ________ @ $31.95 + $4.00 S&H (U.S.) Total Amount: ________ Signature: ____________

Order form on reverse

EDITORS' CHOICE AWARD OS/2 MAGAZINE

C F S Nevada, Inc.

953 E. Sahara Avenue, Suite 9B
Las Vegas, Nevada 89104-3012

Voice	702-732-9616
FAX	702-732-3847
Internet	rrsh@cfsrexx.com
World Wide Web	http://www.cfsrexx.com

Fourth Edition

ISBN: 0-9639854-3-4

SRL & PUBORDER: S246-0078-2