24x7 Scheduler™ 5.3 Multi-platform Edition

JavaScript Reference

Table of Contents

ABOUT THIS REFERENCE 6
CONVENTIONS USED IN THIS DOCUMENT 6
ABBREVIATIONS AND TERMS 6
TRADEMARKS 6
JAVASCRIPT SYNTAX 7
JAVASCRIPT STATEMENTS 7
break 7
comment 7
continue 7
for 7
for…in 8
function 8
if…else 8
return 8
try...catch 9
var 9
while 9
with 9
OPERATORS 11
Operator Precedence 11
JAVASCRIPT OBJECTS 13
Standard Objects 13
Array Object 13
String Object 14
Boolean Object 15
Number Object 15
Math Object 15
Date Object 16
Extension Objects 16
Process Object 16
RunAndWaitInfo Object 16
Directory Object 16
File Object 17
FTP Object 17
CompareInfo Object 17
Mail Object 17
Scheduler Object 17
RESERVED WORDS 17
JAVASCRIPT AUTOMATION EXTENSIONS 19
PROCESS OPERATIONS 19
run 19
runAndWait 19
kill 20
list 20
DIRECTORY OPERATIONS 20
dir 20
remoteDir 21
getWorkDir 21
setWorkDir 21

clean 22
create 22
remove 22
move 23
copyMerge 23
copyReplace 24
exists 24
size 24
zip 25
HIGH-LEVEL FILE OPERATIONS 25
connectFile 25
exists 25
remove 26
rename 26
copy 26
move 27
dateTime 27
save 27
size 28
checksum 28
splitName 29
readAll 29
transfer 29
transferEx 30
unzip 30
zip 31
zipEx 31
LOW-LEVEL FILE OPERATIONS 32
open 32
read 32
write 33
close 33
getPos 33
setPos 34
DATABASE OPERATIONS 35
connect 35
disconnect 35
connectFile 36
execute 37
retrieve 37
exportToFile 38
FTP OPERATIONS 38
appendFile 38
putFile 39
getFile 39
resumeFile 40
renameFile 41
deleteFile 41
fileSize 42
fileExists 42
fileDateTime 43
dir 43
dirCreate 44
dirDelete 44
command 45
config 45
compareDir 48
syncDir 49

MAIL OPERATIONS 50
send 50
sendWithAttachment 50
WEB OPERATIONS 51
getFile 51
postData 52
SCHEDULER OPERATIONS 52
messageBox 52
pause 52
logAddMessage 53
runJob 53
runRemoteJob 54
queueJob 54
queueRemoteJob 54
killJob 55
deleteJob 55
createJob 55
disableJob 56
enableJob 56
setJobProperty 56
getJobProperty 57
findJob 57
getJobs 58
getFolders 58
raiseError 59
stdError 59
stdOutput 59
stdInput 60
exitProcess 60
JOB PROPERTIES IN JDL FORMAT 61
ADDITIONAL JAVA SCRIPT DOCUMENTATION AND EXAMPLES 67

About This Reference

This reference describes JavaScript language and extensions supported in 24x7 Scheduler Multi-platform Edition, an advanced cross-platform job scheduling and automation system. Information in this reference applies to the 24x7 Scheduler version 5.1 running on all supported operation systems. This reference contains information for experienced users of the 24x7 Scheduler and assumes that you have a working knowledge of JavaScript and also understand basic concepts of your operation system.

Conventions Used in This Document

This section describes the style conventions used in this document.

Italic
An *italic* font is used for filenames, URLs, emphasized text, and the first usage of technical terms.

`Monospace`
A `monospaced` font is used for code fragments and data elements.

Bold
A **bold** font is used for important messages, names of options, names of controls and menu items, and keys.

Graphical marks

 **-** This mark is used to indicate product specific options and issues and to mark useful tips.

 **-** This mark is used to indicate important notes.

Abbreviations and Terms

This guide uses common abbreviations for many widely used technical terms including FTP, HTTP, RAS, SQL, DBMS, SSH and other.

Trademarks

JavaScript Syntax

JavaScript Statements

The statements used to control program flow in JavaScript are similar to statements available in Java and C. A statement can span several lines if needed, or several statements can be placed on the same line. A semicolon must be placed between all statements. Since JavaScript is not strict in its formatting, you must provide the line breaks and indentation to make sure the code is readable and easy to understand later.

break

Description: Terminates the current for or while loop and passes control to the first statement after the loop.

Syntax:

```
while (condition)
{
       statements...
       if (condition) break;
       statements...
}
```

comment

Description: Notes from the script author that are ignored by the interpreter. Single line comments are preceded by //. Multiple line comments begin with /* and end with */.

continue

Description: Passes control to the condition in a while loop and to the update expression in a for loop.

for

Description: Creates a loop with three optional expressions enclosed in parentheses and separated by semicolons, followed by a set of statements to be executed during the loop:

Syntax:

```
for( initialExpression; condition; updateExpression)
{
       statements...
}
```

The initial expression is used to initialize the counter variable, which can be a new variable declared with var. The condition expression is evaluated on each pass through the loop. If the condition is true, the loop statements are executed. The update expression is used to increment the counter variable.

for…in

Description: Iterates a variable for all of properties of an object:

Syntax:

```
for (variable in object)
{
       statements...
}
```

For each property, it executes the statement block.

function

Description: Declares a JavaScript function with a name and parameters. To return a value, the function must include a return statement. A function definition cannot be nested within another function.

Syntax:

```
function name ([parameter] [..., parameter])
{
       statements...
}
```

if…else

Description: A conditional statement that executes the first set of statements if the condition is true, and the statements following else if false. If...else statements can be nested to any level.

Syntax:

```
if (condition)
{
       statements...
}
[else
{
       statements...
}]
```

return

Description: Specifies a value to be returned by a function.

Syntax:

```
return expression;
```

try {statements1} [catch (exception){statements2}]

try...catch

Description: The try...catch statement is used to test a block of code for errors. The try block contains the code to be run, while the catch block contains the code to execute if there is an error. The exception argument is a variable in which to store the error.

Syntax:

```
try { ... statement block; } catch (er) { ... error handling block; }
```

Example: The following example attempts to FTP some file and generates an error if FTP fails. This error message is then displayed on the screen.

```
var error = "";

try {
   var server = "my server";
   var user = "test";
   var pass = "1111";
   FTP.deleteFile(server, user, pass, "/pub/1.txt, /pub/2.txt");
}
catch(error) {
  if(error == "Error 1")
    Scheduler.messageBox(error) ;
}
```

var

Description: Declares a variable and optionally initializes it to a value. The scope of a variable is the current function or, when declared outside a function, the current document.

Syntax:

```
var variableName [=value] [..., variableName [=value]]
```

while

Description: Repeats a loop while an expression is true.

Syntax:

```
while (condition)
{
       statements...
}
```

with

Description: Establishes a default object for a set of statements. Any property references without an object are assumed to use the default object.

Syntax:

```
with (object)
{
       statements...
}
```

This statement is especially useful when applied to the Math object for a set of calculations. For example,

Example:

```
with (Math)
{
       var Value1 = cos(angle);
       var Value2 = sin(angle);
}
```

replaces

```
{
       var Value1 = Math.cos(angle);
       var Value2 = Math.sin(angle);
}
```

Operators

JavaScript supports the following operators:

Math

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
-	Negation

Bitwise

Operator	Meaning
&	AND
~	NOT
|	OR
^	exclusive OR
<<	Left shift
>>	Right shift
>>>	Right shift, fill with zeros

Logical

Operator	Meaning
==	Equal
!=	No Equal
<	Less
<=	Less or Equal
>	Greater
>=	Greater or Equal
&&	AND
||	OR
!	NOT
?	Conditional
,	Comma

Operator Precedence

Precedence refers to the order in which compound operations are computed. Operators on the same level have equal precedence. Calculations are computed from left to right on all binary operations beginning with the operators at the top of the list and working down.

Call, member	.	[ ]	( )		
negation/increment	++	-	!	~	-
multiply/divide	*	/	%		
addition/subtraction	+	-			
shift	<<	>>	>>>		
relational	<	>	<=	>=	
equality	==	!=			
bitwise AND	&				
bitwise XOR	^				
bitwise OR	|				
logical AND	&&				
logical OR	||				
conditional	?:				
assignment	=	op=			
comma	,				

JavaScript Objects

JavaScript is an object-oriented language, and as such, includes a set of built-in objects to represent the HTML document, especially form elements. Built-in objects can be accessed by both the client and server.

Standard Objects

24x7 Scheduler Multi-platform Edition supports the following built-in standard JavaScript objects:

Array Object

Description: Contains an array of objects or values.

Array object supports the following properties and functions:

length	A read/write property indicating the current number of elements within the array. You may set this property to dynamically expand an array's length.
concat(val1, ...)	Concatenates all the argument values to the existing array, and returns the new array. Values can be another array.
join([separator])	Converts each element within the array to a string, and joins them into one large string. Pass in an optional separator as argument to be used to separate each array element. If none is passed, the default comma (') is used.
pop( )	Deletes the last element within array and returns the deleted element. Original array is modified.
Push(…, ...)	Adds the argument values to the end of the array, and modifies the original array with the new additions. Returns the new length of the array.
Reverse( )	Reverses the order of all elements within the array. Original array is modified.
shift()	Deletes and returns the first element within the array. Original array is modified to account for the missing element (so 2nd element now becomes the first etc).
slice(start, [end])	Returns a "slice" of the original array based on the start and end arguments. The slice includes the new array referenced by the start index and up to but NOT including the end index itself. If "end" is not specified, the end of the array is assumed.
splice(startIndex, [how_many], [value1, ...])	Deletes array elements starting from startIndex, and replaces them with value1, value2 etc. Returns the elements deleted from array.
sort([SortFunction])	Sorts an array alphabetically and ascending. By passing in an optional Sort Function, you may sort numerically and by other criteria as well.
toSource()	Returns an array literal representing the specified array.
toString()	Returns a string representing the array and its elements.
unshift(value1, ...)	Adds the argument values to the beginning of the array, pushing existing arrays back. Returns the new length of the array .Original array is modified.
valueOf()	Returns the primitive value of the array.
shift()	Deletes and returns the first element within the array. Original array is modified to account for the missing element (so 2nd element now becomes the first etc).
slice(start, [end])	Returns a "slice" of the original array based on the start and end arguments. The slice includes the new array referenced by the start index and up to but NOT including the end index itself. If "end" is not specified, the end of the array is assumed.
splice(startIndex,	Deletes how_many array elements starting from startIndex, and replaces

[how_many], [value1, ...])	them with value1, value2 etc. Returns the elements deleted from array.

String Object

Description: Contains a string of characters.

String object supports the following properties and functions:

length	This property returns the length of the string (# of characters).
charAt(x)	Returns the character at the "x" position within the string.
charCodeAt(x)	Returns the Unicode value of the character at position "x" within the string.
concat(v1, v2,...)	Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.
fromCharCode(c1, c2,...)	Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance. For example: String.fromCharCode().
indexOf(substr, [start])	Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is 0.
lastIndexOf(substr, [start])	Searches and (if found) returns the index number of the searched character or substring within the string. Searches the string from end to beginning. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is string.length-1.
match(regexp)	Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match is found.
replace(regexp, replacetext)	Searches and replaces the regular expression portion (match) with the replaced text instead.
search(regexp)	Tests for a match in a string. It returns the index of the match, or -1 if not found.
slice(start, [end])	Returns a substring of the string based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.
split(delimiter, [limit])	Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return.
substr(start, [length])	Returns the characters in a string beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed.
substring(from, [to])	Returns the characters in a string between "from" and "to" indexes, NOT including "to" "To" is optional, and if omitted, up to the end of the string is assumed.
toLowerCase()	Returns the string with all of its characters converted to lowercase.
toUpperCase()	Returns the string with all of its characters converted to uppercase.

Boolean Object

Description: This object is used to turn a value that is not boolean into a value that is boolean which is true or false.

JavaScript supports the following functions of the Boolean object:

toString()	Returns a string specifying the value of the Boolean, in this case, "true" or "false."
valueOf()	Returns the primitive value of a Boolean object.

Number Object

Description: The number object must have an instance created in order to use it. The number object has no specific functions. The Number object exposes the following properties:

MAX_VALUE - The largest value that may be used in JavaScript.

MIN_VALUE - The smallest value that may be used in JavaScript.

NaN - Used to indicate a value is not a number. A number object may be set to this value to indicate that it is not really a number, for example:

```
if (Month < 1 || Month > 12)
{
       Month = Number.NaN;
}
```

NEGATIVE_INFINITY - The value returned if a negative overflow occurs. Any numeric value divided by this is 0.

POSITIVE_INFINITY - The value returned if a positive value overflow occurs. Any numeric value divided by this is 0.

prototype - For creating more properties.

Math Object

Description: Provides numerical constants and mathematical functions.

JavaScript supports the following mathematical functions (methods of the Math object):

Math.abs(a)	the absolute value of a
Math.acos(a)	arc cosine of a
Math.asin(a)	arc sine of a
Math.atan(a)	arc tangent of a
Math.atan2(a,b)	arc tangent of a/b
Math.ceil(a)	integer closest to a and not less than a
Math.cos(a)	cosine of a
Math.exp(a)	exponent of a
Math.floor(a)	integer closest to and not greater than a
Math.log(a)	log of a base e
Math.max(a,b)	the maximum of a and b

Math.min(a,b)	the minimum of a and b
Math.pow(a,b)	a to the power b
Math.random()	pseudorandom number in the range 0 to 1
Math.round(a)	integer closest to a
Math.sin(a)	sine of a
Math.sqrt(a)	square root of a
Math.tan(a)	tangent of a

Date Object

Description: Stores a date in the number of milliseconds since 1/1/1970, 00:00:00, and returns a date string in the format "Thu, 11 Jan 1996 06:20:00 GMT".

Date object supports the following constructors:

Date()	Use the current date and time to create an instance of the object date.
Date(dateString)	Use the date specified by the string to create the instance of the date object. String format is "month day, year hours:minutes:seconds".
Date(year, month, day)	Create an instance of date with the specified values. Year is 0 to 99.
Date(year, month, day, hours, minutes, seconds)	Create an instance of date with the specified values.

Extension Objects

24x7 Scheduler Multi-platform Edition extends standard set of JavaScript objects with additional objects specific to scheduler operations and process management. All these functions are described in detail in the JavaScript Automation Extensions topic. The following extension objects are supported:

Process Object

Description: Provides functions for starting, stopping and listing system processes. For more information see Process Operations topic.

RunAndWaitInfo Object

Description: Helper object provides additional functions and attributed in process management operations. For more information see runAndWait topic.

Directory Object

Description: Provides functions for manipulating file directories and listing directory contents. For more information see Directory Operations topic.

File Object

Description: Provides functions and for manipulating files and reporting file attributes, including file creating, update, deletion and many other. For more information see High-Level File Operations and Low-Level File Operations topics.

FTP Object

Description: Provides functions for performing common FTP operations also functions for replicating files over FTP connections. For more information see FTP Operations topic.

CompareInfo Object

Description: Helper object provides additional functions and attributed for FTP operations. For more information see compareDir topic.

Mail Object

Description: Provides functions for sending email messages including messages with file attachments. For more information see Mail Operations topic.

Scheduler Object

Description: Provides many functions for starting and stopping jobs, manipulating job properties, writing messages to the system job log, and other. For more information see Scheduler Operations topic.

Reserved Words

The following words cannot be used as user objects or variables in coding JavaScript. Not all are currently in use by JavaScript-they are reserved for future use.

abstract	for	public
boolean	function	return
break	goto	short
byte	if	Static
case	implements	super
catch	import	Switch
char	in	synchronize
const	instanceOf	this

continue	int	throw
default	interface	Throws
do	long	transient
double	native	true
else	new	try
extends	null	var
false	package	void
final	private	while
finally	protected	with
float		

JavaScript Automation Extensions

24x7 Scheduler provides a predefined set of embedded JavaScript automation objects and functions. To access these functions you can use already defined global JavaScript variables ‘File’, ’Process’, ’Directory’, ‘Scheduler’. For example,

```
var pid = Process.run("notepad");
Scheduler.messageBox("pid: " + pid);
```

This job will run notepad application and show message box with the internal process ID.

Process Operations

run

Prototype: int Process.run(String command)

Description: Runs the specified program or command.

Parameters:
command – is a string whose value is the full or partial path and filename of a shell command or other executable file.

Return: returns the internal process ID. This ID can be used only in this job scope to kill the created process, for example.

Example:

```
var pid = Process.run("/bin/cp /home/trade/db.dat /home/backup");
```

runAndWait

Prototype: RunAndWaitInfo Process.runAndWait(String command, int timeout)

Description: Runs the specified program or command and enters an efficient wait state until this process finishes or the timeout interval elapses. In the latter case, the 24x7 Scheduler forcedly terminates the process.

Parameters:

command – is a string whose value is the full or partial path and filename of a shell command or other executable file.

timeout – A number of milliseconds whose value is the maximum time interval within which you allow the specified process to run. Use 0 timeout to allow infinite waiting.

Return: returns **RunAndWaitInfo** structure that has the following functions:

String getOutput() – returns a string variable that receives the data written by the created process to the standard error and standard output.

int getProcessId() – returns the internal process ID.

Example:

```
var runInfo = Process.runAndWait("/bin/ls", 0);
Scheduler.messageBox("output: " + runInfo.getOutput());
```

kill

Prototype: Process.kill(int pid)

Description: Terminates process by the internal pid.

Parameters:

pid – the internal pid of the process to kill.

Example:

```
var pid = Process.run("/bin/sleep 60");
if (Scheduler.messageBox('Do you want to kill the program?'))
{
  Process.kill(pid);
}
```

list

Prototype: String Process.list()

Description: Returns list of operation system processes including process IDs and names. Do not confuse system process IDs and names with IDs and names of 24x7 Scheduler jobs and internal processes.

Return: Returns process list as a string. Each process is separated by a new line

Example:

```
var processList = Process.list()
Scheduler.messageBox(processList);
```

Directory Operations

dir

Prototype: String Directory.dir(String fileMask)

Description: Returns comma-separated list of files in the current working directory.

Parameters:

fileMask - a string whose value is the file mask to use for searching (* - any word, ? – any symbol).

Return: Returns comma-separated list of files in the current working directory.

Example:

```
var files = Directory.dir("*.log");
Scheduler.messageBox("Log files: " + files);
```

remoteDir

Prototype: String Dir.remoteDir(String agentName, String fileMask, String user, String password)

Description: Returns comma-separated list of files in the specified directory on the remote computer.

Parameters:

agentName - a string whose value is the name of the remote agent profile to use for remote computer connection.

fileMask - a string whose value is the file mask to use for searching (* - any word, ? – any symbol).

user - a string whose value is the user name for authentication on the remote system.

Password - a string whose value is the user password for authentication on the remote system.

Return: Comma-separated list of files in the remote directory.

Example:

```
var files = Directory.remoteDir("agent DB2 server",
                                "/home/db2/logs/*.log",
                                "oscar", "secret");
Scheduler.messageBox("Log files: " + files);
```

getWorkDir

Prototype: String Directory.getWorkDir()

Description: Reports name of the current working directory. The default working directory is the 24x7 Scheduler installation directory.

Return: Returns full path to the working directory.

Example:

```
var dir = Directory.getWorkDir();
Scheduler.messageBox("Working directory: " + dir);
```

setWorkDir

Prototype: Directory.setWorkDir(String path)

Description: Changes current working directory. Each JavaScript job session runs in its own environment and has session logical working directory. Setting of working directory affects Directory.dir, Directory.getWorkDir, Process.run, Process.runAndWait functions. It does not affect high-level or low-level operations.

Parameters:

path - a string whose value is the full or relative path to the new working directory.

Example:

```
Directory.setWorkDir("/home/test");
var dir = Directory.getWorkDir();
Scheduler.messageBox("Working directory: " + dir);
```

clean

Prototype: void Directory.clean(String path)

Description: Deletes all files and subdirectories recursively in the specified directory.

Parameters:

path - A string whose value is the full or relative path to the directory.

Example:

```
Directory.clean("/home/test");
```

create

Prototype: void Directory.create(String path)

Description: Creates the specified directory and recursively all subdirectories in the specified path.

Parameters:

path - A string whose value is the full or relative path to the directory.

Example:

```
Directory.create("/home/test/dir1/dir2/dir3");
```

remove

Prototype: void Directory.remove(String path)

Description: Removes the specified directory and recursively all files and subdirectories in the specified path.

Parameters:

path - A string whose value is the full or relative path to the directory.

Example:

```
Directory.remove("/home/test/dir1 ");
```

move

Prototype: void Directory.move(String sourcePath, String targetPath)

Description: Moves the specified directory and all its content to a different location. The new location can be on the same or different drive, volume, and/or file-system. If the target path already exists and non-empty, its content is replaced with the moved files and directories.

Parameters:

sourcePath - A string whose value is the full or relative path to the directory to move.

targetPath - A string whose value is the full or relative path to the new location.

Example:

```
Directory.move("C:\\data\\dir1", "D:\\archive");
```

copyMerge

Prototype: void Directory.copyMerge(String sourcePath, String targetPath)

Description: Copies the specified directory and all its content to a different location. The target location can be on the same or different drive, volume, and/or file-system. If the target path doesn't exist, it is created. If the target path already exists, the content is merged - files from the source directory overwrite files with the same names in the target directory; files that exist in the target directory only, remain in that directory.

Parameters:

sourcePath - A string whose value is the full or relative path to the directory to copy.

targetPath - A string whose value is the full or relative path to the target location.

Example:

```
Directory.copyMerge("C:\\data\\dir1", "D:\\archive");
```

copyReplace

Prototype: void Directory.copyReplace(String sourcePath, String targetPath)

Description: Copies the specified directory and all its content to a different location. The target location can be on the same or different drive, volume, and/or file-system. If the target path doesn't exist, it is created. If the target path already exists, the content is replaced with files and directories from the target directory.

Parameters:

sourcePath - A string whose value is the full or relative path to the directory to copy.

targetPath - A string whose value is the full or relative path to the target location.

Example:

```
Directory.copyReplace("C:\\data\\dir1", "D:\\archive");
```

exists

Prototype: boolean Directory.exists(String dirName)

Description: Tests for existence of the specified directory.

Parameters:

dirName - A string whose value is the full or relative path to the directory to test.

Return: Returns TRUE if file exists or FALSE otherwise.

Example:

```
var dirFound = Directory.exists("/var/logs/load");
if (dirFound ) Scheduler.messageBox("Directory found");
```

size

Prototype: long Directory.size(String path)

Description: Reports total size of all files in the specified directory.

Parameters:

path - A string whose value is the full or relative path to the directory.

Return: Returns total size of all files.

Example:

```
var totalSize = Directory.size("/var/messages");
```

zip

Prototype: void Directory.zip(String zipName, String path)

Description: Zips all files and recursively all directories in the specified path into a ZIP file.

Parameters:

zipName - A string whose value is the name of the target zip file.

path - A string whose value is the full or relative path to the directory.

Example:

```
Directory.zip("/home/archive/latest.zip", "/home/incoming/transactions");
Directory.clean("/home/incoming/transactions ");
```

High-Level File Operations

connectFile

Description: This high-level file operation is used to open Microsoft Excel and Microsoft Access files for read/write operations and work with them using SQL queries as if Excel worksheets were database systems containing tables. For more information see connectFile topic in the Database Operations chapter.

exists

Prototype: boolean File.exists(String fileName)

Description: Checks if the specified file exists.

Parameters:

fileName - a string whose value is the name of the file that you want to check.

Return: Returns TRUE if file exists or FALSE otherwise

Example:

```
var fileFound = File.exists("/var/mail/message.txt");
if (fileFound ) Scheduler.messageBox("File found");
```

remove

Prototype: File.remove(String fileName)

Description: Deletes the specified file. The file must exist.

Parameters:

fileName - a string whose value is the name of the file that you want to delete.

Example:

```
File.remove("/var/mail/message.txt");
Scheduler.messageBox("File has been deleted");
```

rename

Prototype: File.rename(String fileOldName, String fileNewName)

Description: Renames the specified file. The file must exist.

Parameters:

fileOldName - a string whose value is the name of the file that you want to rename.

fileNewName - a string whose value is the new name.

Example:

```
File.rename("/var/mail/message.txt", "/var/mail/message.old");
Scheduler.messageBox("File has been renamed");
```

copy

Prototype: File.copy(String sourceFileName, String targetFileName)

Description: Copies the specified file. The file must exist. The new file can be created in the same or different directory.

Parameters:

sourceFileName - a string whose value is the name of the file that you want to copy.

targetFileName - a string whose value is the name of the target file.

Example:

```
File.copy("/var/mail/message.txt", "/var/mail/message.bak");
Scheduler.messageBox("File has been copied");
```

move

Prototype: File.nove(String sourceFileName, String targetFileName)

Description: Moves the specified file. The file must exist. The old file is copied first and if the copy operation success the old file is deleted.

Parameters:

sourceFileName - a string whose value is the name of the file that you want to move.

targetFileName - a string whose value is the name of the target file.

Example:

```
File.move("/var/mail/message.txt", "/var/mail.bak/message.txt");
Scheduler.messageBox("File has been moved");
```

dateTime

Prototype: Date File.dateTime(String fileName)

Description: Reports the date and time that a file was last modified.

Parameters:

filename - a string whose value is the file name.

Return: Returns Date object with the following functions:

int getYear() – returns year.

int getMonth() – returns month.

int getDate() – retunrs day.

int getHour() – returns hour.

int getMinute() – returns minute.

int getSecond() – returns second.

Example:

```
var dateTime = File.dateTime('/var/log/messages');
Scheduler.messageBox(dateTime.getYear() + "." + dateTime.getMonth() + "." +
                     dateTime.getDate() + " " + dateTime.getHour() + ":" +
                     dateTime.getMinute() + ":" + dateTime.getSecond());
```

save

Prototype: File.save(String fileName, String text)

Description: Saves text data in the specified file.

Parameters:

fileName - a string variable whose value is the file name into which you want to save the text.

text - a string whose value is the data want to save in the file.

Example:

```
var text = "The first line\nThe second line";
File.save("/home/scheduler/test.txt", text);
```

size

Prototype: long File.size(String fileName)

Description: Reports the length of a file in bytes.

Parameters:

fileName - a string whose value is the name of the file whose length you want to obtain. If file is not located in the current working directory, you must specify the fully qualified file name.

Return: Returns length of the file.

Example:

```
var fileSize = File.size("/var/log/messages");
Scheduler.messageBox("log file size: " + fileSize);
```

checksum

Prototype: long File.checksum(String fileName)

Description: Reports file CRC checksum.

Parameters:

fileName - A string whose value is the name of the file.

Return: Returns CRC checksum of the file.

Example:

```
var fileCRC = File.checksum("/var/log/messages");
Scheduler.messageBox("Checksum: " + fileCRC);
```

splitName

Prototype: String[2] File.splitName(String fileFullName)

Description: Separates file path part and file name part for a given full file name or file mask.

Parameters:

fileName - a string whose value is the full or partial file name.

Return: Returns string array containing 2 elements: element with index 0 contains file path, element with index 1 contains file name.

Example:

```
var parts = File.splitName("/home/data/archive/file1.gz");
Scheduler.messageBox("File path: " + parts[0]);
Scheduler.messageBox("File name: " + parts[1]);
```

readAll

Prototype: String File.readAll(String fileName)

Description: Loads entire file contents into script variable.

Parameters:

fileName - a string whose value is the name of the file that you want to read.

Return: Returns the buffer with file contents

Example:

```
var content = File.readAll("/var/mail/message.txt");
Scheduler.messageBox(content);
```

transfer

Prototype: void File.transfer(String agentName, String sourceFile, String targetFile, String user, String password)

Description: Copies the specified file from local to remote system. The file is automatically compressed and encrypted on the local system, and then after transmission it is automatically decrypted and decompressed on the remote system.

Parameters:

agentName - a string whose value is name of the remote agent profile to connect to.

sourceFile – a string whose value is the name of the file on the local system of network share to transfer.

targetFile - a string whose value is the name of the file on the remote system.

user – a string whose value is the user name for authentication on the remote system.

password – a string whose value is the user password for authentication on the remote system.

Return: None

Example:

```
File.transfer("RemoteServer", "c:\\data\\data01.csv",
              "/home/oracle/data/data01.csv", "oscar" , "secret");
```

transferEx

Prototype: void File.transferEx(String agentName, String direction, String[] sourceFiles, String[] targetFiles, String user, String password)

Description: Copies the specified files from local to remote system or from remote to local. The files are automatically compressed and encrypted on the source system, and then after transmission, they are automatically decrypted and decompressed on the target system.

Parameters:

agentName - a string whose value is name of the remote agent profile to connect to.

direction - a string constant specifying direction in which to transfer the specified files. The value must be either FromRemote or ToRemote.

SourceFiles[ ] – a string array of names of files to transfer."/>

TargetFiles[ ] – a string array of names of files to be updated or created on the target system. The number of files in this array must be the same as the number of files in the sourceFiles array. File names and locations can differ from the source.

user – a string whose value is the user name for authentication on the remote system.

password – a string whose value is the user password for authentication on the remote system.

Return: None

Example:

```
var srcFiles = new Array();
srcFiles[0] = "/home/oracle/data/data01.csv";
srcFiles[1] = "/home/oracle/data04/format.txt";
var dstFiles = new Array();
dstFiles[0] = "c:\\data\\data01.csv";
dstFiles[1] = "c:\\data\\format.txt";

File.transferEx("RemoteServer", "FromRemote", srcFiles, dstFiles,
              "oscar" , "secret");
```

unzip

Prototype: void File.unzip(String zipName, String destDir)

Description: Unzips files from the specified ZIP archive. The specified ZIP file can have any extension but must be in the standard ZIP format.

Parameters:

zipName - a string whose value is the name of the zip file to unzip.

destFile – a string whose value is the name of the file on the local system of network share to transfer.

Return: None

Example:

```
File.unzip( "c:\\data\\data01.zip", "c:\\data\\unzipped" );
```

zip

Prototype: void File.zip(String zipName, String files)

Description: Zips one or multiple files to the specified ZIP file.

Parameters:

zipName - a string whose value is the name of the target zip file.

files – a string whose value is the comma-separated list of names of files to zip.

Return: None

Example:

```
File.zip( "c:\\backup\\data@T"yyyymmdd".zip",
       "c:\\data\\file1,c:\\data\\file2,c:\\data\\file3" );
```

zipEx

Prototype: void File.zipEx(String zipName, String[] files)

Description: Zips one or multiple files to the specified ZIP file.

Parameters:

zipName - a string whose value is the name of the target zip file.

files[ ] – a string array of names of files to zip.

Return: None

Example:

```
var srcFiles = new Array();
srcFiles[0] = "/home/oracle/data/data01.csv";
srcFiles[1] = "/home/oracle/data04/data02.csv";
srcFiles[2] = "/home/oracle/data04/format.txt";
File.zipEx( "/home/archive/data@T"yyyymmdd".zip", srcFiles );
```

Low-Level File Operations

open

Prototype: int File.open(String fileName, String fileAccess, boolean append)

Description: Opens the specified file for reading or writing and assigns it a unique file number. You use this number to identify the file when you read, write, or close the file.

Parameters:

fileName - a string whose value is the name of the file you want to open. If fileName is not in the operating system's search path, you must enter the fully qualified name.

fileAccess – a string constant whose value specifies whether the file is opened for reading or writing. Values are:

- "Read" - Read-only access
- "Write" - Write-only access
- "ReadWrite" - Both read and write access

append - A boolean whose value specifies whether existing data in the file is overwritten when file is opened for write operation. ‘append’ is ignored if the fileAccess argument is "Read". Values are:

- True - Write data to the end of the file.
- False - Replace all existing data in the file.

Return: Returns the internal file number assigned to the opened file.

Example:

```
var data = "1234567890";
File.save("test.txt", data);
var fileNumber = File.open("test.txt", "Read", false);
var read = File.read(fileNumber, 4);
Scheduler.messageBox("read 4 symbols: " + read);
File.close(fileNumber);
```

read

Prototype: String File.read(int fileNumber, int bytes)

Description: Reads data from the file associated with the specified file number, which was assigned to the file with the File.open operation.

Parameters:

fileNumber - a number whose value is the file number previously assigned to the file when it was opened by File.open operation.

Bytes - a number whose value indicates how many bytes you want to read from the file.

Return: returns a string variable with the read data

Example:

```
var data = "1234567890";
```

```
File.save("test.txt", data);
var fileNumber = File.open("test.txt", "Read", false);
var read = File.read(fileNumber, 4);
Scheduler.messageBox("read 4 symbols: " + read);
File.close(fileNumber);
```

write

Prototype: File.write(int fileNumber, String data)

Description: Writes data to the file associated with the specified file number, which was assigned to the file with the File.open operation.

Parameters:

fileNumber a number whose value is the file number previously assigned to the file when it was opened by File.open operation.

data - a string whose value is the text that you want to write to the file.

Example:

```
var data = "1234567890";
File.save("test.txt", data);
var fileNumber = File.open("test.txt", "ReadWrite", true);
File.write(fileNumber, "_appended_string");
var read = File.readAll("test.txt");
Scheduler.messageBox("modified file: " + read);
File.close(fileNumber);
```

close

Prototype: File.close(int fileNumber)

Description: Closes the file associated with the specified file number. The file number was assigned to the file with the File.open operation.

Parameters:

fileNumber - the number assigned to the file you want to close. The File.open operation returns the file number when it opens the file.

Example:

```
var fileNumber = File.open("log.txt", "ReadWrite", true);
File.write(fileNumber, "log message");
File.close(fileNumber);
```

getPos

Prototype: int File.getPos(int fileNumber)

Description: Reports current position in the specified file previously opened by File.open operation.

Parameters:

fileNumber - the file number previously assigned to the file when it was opened by File.open operation.

Return: returns the file position after the read/write operation or zero if no operation has been performed after file opening.

Example:

```
var data = "1234567890";
File.save("test.txt", data);
var fileNumber = File.open("test.txt", "ReadWrite", true);
File.write(fileNumber, "_appended_string");
var filePos = File.getPos(fileNumber);
Scheduler.messageBox("Current file position: " + filePos);
File.close(fileNumber);
```

setPos

Prototype: File.setPos(int fileNumber, int pos, String origin)

Description: Moves the file pointer to the specified position in a file previously opened by File.open operation. The file pointer is the position in the file at which the next read or write begins.

Parameters:

fileNumber - the file number previously assigned to the file when it was opened by File.open operation.

pos – numeric position to be set

origin - a string constant whose value specifies from where you want to set the position. Values are:

- "START" - At the beginning of the file.
- "CURRENT" - At the current position.
- "END" - At the end of the file.

Example:

```
var data = "1234567890";
File.save("test.txt", data);
var fileNumber = File.open("test.txt", "ReadWrite", false);
File.setPos(fileNumber, 5, "START");
File.write(fileNumber, "**");
var read = File.readAll("test.txt");
Scheduler.messageBox("modified file: " + read);
File.close(fileNumber);
```

Database Operations

connect

Prototype: void Database.connect(String profile)

Description: Establishes database connection using the specified database profile.

 **Important Notes:**

- Database.connect method must be executed before other database actions can be processed.
- One job may have only one database connection open at a time. However, multiple jobs may have multiple database connections opened simultaneously. The same job can also open and close multiple connections sequentially.
- All other database methods executed after Database.connect are sent to the database specified in the profile.
- You are responsible for closing database connections. Failure to close connections may lead to resource leaks. Use Database.disconnect method to close previously opened connection.

Parameters:

profile - A string whose value is the name of the profile defined in system settings.

Examples:

```
// connect to database server and execute stored procedure
Database.connect( "PRODUCTION" );
Database.execute( "CALL MySchema.MyStoredProcedure('param1', 'param')" );
//retrieve some values from the database
var newVal = Database.retrieve(  "SELECT * FROM MySchema.NyTable " +
                                 "WHERE date_idx = trunc(sysdate)");
// disconnect from database server and save results in a local file
Database.disconnect();
File.save( "my_file.txt", newVal );
```

disconnect

Prototype: void Database.disconnect( )

Description: Disconnects from the database server.

Parameters:

None – this method has no parameters.

Examples:

```
// connect to database server and execute stored procedure
   Database.connect( "PRODUCTION" );
   Database.execute( "CALL MySchema.MyStoredProcedure('param1', 'param')" );
   //retrieve some values from the database
   var newVal = Database.retrieve(  "SELECT * FROM MySchema.NyTable " +
                                    "WHERE date_idx = trunc(sysdate)");
   // disconnect from database server and save results in a local file
```

```
Database.disconnect( );
File.save( "my_file.txt", newVal );
```

connectFile

Prototype: void Database.connectFile(String fileName, String fileType)

Description: Establishes file-based database connection using the specified file name and type. **This method can be used with Microsoft Excel and Microsoft Access files only**. This method can be called on Windows based systems only. The Excel and Access interfaces implemented in 24x7 depend on the availability of Excel and Access database drivers, which are pre-installed by default on all Windows systems. Refer to Microsoft documentation for the functions and SQL command syntax supported by the version of the driver(s) installed on your system.

 **Important Notes:**

- Database.connectFile method can be used on Windows based systems only. be executed before other database actions can be processed.
- Database.connectFile method must be executed before other database actions can be processed.
- One job may have only one database connection open at a time. However, multiple jobs may have multiple database connections opened simultaneously. The same job can also open and close multiple connections sequentially.
- All other database methods executed after Database.connectFile are sent to the database specified in the profile.
- You are responsible for closing database connections. Failure to close connections may lead to resource leaks. Use Database.disconnect method to close previously opened connection.

Parameters:

fileName - A string whose value is the fully qualified name of the file to connect to.

fileType - A string whose value is the type of the file to connect to. This parameter controls type of the database driver the script engine will attempt to load and use for the following database operations. The supported values are:

- Excel
- Access

Examples:

```
// connect to Excel file and retrieve data from worksheet Transactions
// for a given store
Database.connectFile( "C:\\ExlFiles\\Trans\\August\\Store1.xls", "Excel" );
//retrieve records for August 15
var records =       Database.retrieve(  "SELECT * FROM [Transactions] " +
                                        "WHERE [Store Name] = 'Store 1'");
// disconnect from the Excel file and save results in a local text file
Database.disconnect( );
File.save( "my_file.txt", records );
```

execute

Prototype: int Database.execute(String sql)

Description: Executes SQL statement that does not produce a result set.

Parameters:

sql - A string whose value is a valid SQL command that you want to send to the database.

Return: Returns number of records affected by the executed database command. This value is reported by the database server and makes sense only for commands updating, inserting or deleting data in the database.

Examples:

```
// connect to database server and execute stored procedure
Database.connect( "PRODUCTION" );
Database.execute( "CALL MySchema.MyStoredProcedure('param1', 'param')" );
// disconnect from the database
Database.disconnect( );
```

retrieve

Prototype: String Database.retrieve(String sql)

Description: Retrieves data from the database.

 **Important Notes:** Do not use this command to retrieve large volumes of data because the returned data is stored in memory. If you need to process large data volumes, use Database.export method.

Parameters:

sql - A string whose value is a valid SQL command that you want to send to the database and expert to return results. The command could be specified a in a form of SELECT statement, a database stored procedure call, or a SQL batch, if supported by your database server.

Return: Returns result set as a tab-separated multi-line data value or a single data value, in case the command returns only a single value.

 **Note:** NULL values are returned as empty strings. All non-string values are converted to string equivalents using default data conversion rules for the system running the scheduler and may depend on your local regional settings.

Examples:

```
// connect to database server and execute stored procedure
Database.connect( "PRODUCTION" );
var someValue = Database.retrieve("SELECT max(ColX) FROM MySchema.NyTable " +
                                  "WHERE date_idx = trunc(sysdate)");
// disconnect from the database
Database.disconnect( );
```

exportToFile

Prototype: void Database.exportToFile(String command, string localFile *[, String separator]*)

Description: Retrieves data from the database and saves it into a local file.

Parameters:

command - A string whose value is either a valid SQL command that you want to send to the database and expert to return results or a name of a database table or view. The command could be a in a form of SELECT statement, database stored procedure call or a SQL batch. In case the specified command consist of a single word, it is assumed to be a table or view name and the required SQL query is constructed automatically using this name.

localFile - A string whose value is the name of the file into which you want to write the exported data.

separator - This optional parameter can be used to specify column separator symbol(s). If not specified, tab character is used by default.

Return: Returns number of records exported.

Examples:

```
// connect to database server and execute stored procedure
Database.connect( "PRODUCTION" );
//retrieve values from the database using complete query
Database.exportToFile( "SELECT * FROM MyTable WHERE colA = 1 ORDER BY colB",
                "/home/data_exports/data.txt" );
//retrieve values from the database using table name
Database.exportToFile( "MyTable", "/home/data_exports/mytable_data.txt" );
// disconnect from the database
Database.disconnect( );
```

FTP Operations

appendFile

Prototype: FTP.appendFile(String server, String user, String password, String source, String target)

Description: Transfers a file from local system to the specified remote FTP server and stores it under the specified file name, creating a new remote file in the process or appending data to an existing remote file.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

source - A string whose value is the name of the file to transfer from the local system

target - A string whose value is the name of the file to create on the remote system. Both source and target file can be either partially or fully qualified file names relative to the current directory.

Note: To transfer multiple files in one pass, specify the source files as a comma separated list. The target files must be also specified as a comma separated list. Make sure to specify the same number of file names in the source and target file lists.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.appendFile(server, user, pass, "c:\\1.txt, c:\\2.txt",
                                   "/pub/1.txt, /pub/2.txt");
```

putFile

Prototype: FTP.putFile(String server, String user, String password, String source, String target)

Description: Transfers a file from local system to the specified remote FTP server and stores it under the specified file name, creating a new remote file in the process.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

source - A string whose value is the name of the file to transfer from the local system

target - A string whose value is the name of the file to create on the remote system.

Note: To transfer multiple files in one pass, specify the source files as a comma separated list. The target files must be also specified as a comma separated list. Make sure to specify the same number of file names in the source and target file lists.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.putFile(server, user, pass, "c:\\1.txt", "/pub/1.txt");
```

getFile

Prototype: FTP.getFile(String server, String user, String password, String source, String target)

Description: Retrieves a file from the specified FTP server and stores it under the specified file name, creating a new local file in the process.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

source - A string whose value is the name of the file to retrieve from the remote system.

target - A string whose value is the name of the file to create on the local system.

Note: To transfer multiple files in one pass, specify the source files as a comma separated list. The target files must be also specified as a comma separated list. Make sure to specify the same number of file names in the source and target file lists.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.getFile(server, user, pass, "1.txt", "e:\\1.txt");
FTP.getFile(server, user, pass, "/pub/INSTALL", "e:\\INSTALL");
```

resumeFile

Prototype: FTP.resumeFile(String server, String user, String password, String source, String target)

Description: Retrieves a file from the specified FTP server and stores it under the specified file name, creating a new local file in the process or appending to local file if it already exists. FTP.resumeFile statement is identical to FTP.getFile statement except that it attempts to resume broken downloads or perform incremental downloads of files whose size have increased since the last download.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

source - A string whose value is the name of the file to retrieve from the remote system.

target - A string whose value is the name of the file to create on the local system.

Note: To transfer multiple files in one pass, specify the source files as a comma separated list. The target files must be also specified as a comma separated list. Make sure to specify the same number of file names in the source and target file lists.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.resumeFile(server, user, pass, "1.txt", "e:\\1.txt");
```

renameFile

Prototype: FTP.renameFile(String server, String user, String password, String oldname, String newname)

Description: Renames the specified remote file on the specified FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

oldname - A string whose value is the name of the file to rename.

newname - A string whose value is the new name of the file.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.renameFile(server, user, pass, "/pub/1.txt", "/pub/2.txt");
```

deleteFile

Prototype: FTP.deleteFile(String server, String user, String password, String file)

Description: Renames the specified remote file on the specified FTP server..

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

file- A string whose value is the name of the file that you want to delete.

Note: FTP.deleteFile statement can delete multiple files in one pass. This is more efficient than calling FTP.deleteFile for each file separately, which requires a separate FTP connection for every file. To delete multiple files in one pass, specify multiple files names in the file parameter as a comma separated list.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.deleteFile(server, user, pass, "/pub/1.txt, /pub/2.txt");
```

fileSize

Prototype: int FTP.fileSize(String server, String user, String password, String file)

Description: Reports size of the specified file on the specified FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

file - A string whose value is the name of the file that you want to check.

Return: Returns file size in bytes (-1 if file not found).

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
size = FTP.fileSize(server, user, pass, "/pub/1.txt");
Scheduler.messageBox(size);
```

fileExists

Prototype: boolean FTP.fileExists(String server, String user, String password, String file)

Description: Reports whether the specified file exists on the specified FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

file- A string whose value is the name of the file that you want to check.

Return: Returns TRUE if file exists, and FALSE otherwise.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
if(FTP.fileExists(server, user, pass, "/pub/README"))
       Scheduler.messageBox("Found README file in the pub directory!");
```

fileDateTime

Prototype: Date FTP.fileDateTime(String server, String user, String password, String file)

Description: Reports date and time of the specified file on the specified FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

file - A string whose value is the name of the file that you want to check.

Return: Returns Date object. You can use the following functions to obtain date and time parts:

int getYear()
int getMonth()
int getDate()
int getHour()
int getMinute()
int getSecond()

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
Date d = FTP.fileDateTime(server, user, pass, "/pub/1.txt");
var str = "Date is " + d.getMonth() + "/" + d.getDate() + "/" +
       d.getYear() + "/" + d.getHour() + ":" + d.getMinute() + ":" +
       d.getSecond();
Scheduler.messageBox(str);
```

dir

Prototype: String FTP.dir(String server, String user, String password, String fileMask)

Description: Returns comma-separated list of files in the specified directory on the specified FTP server

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

fileMask – a string whose value is the file mask that you want to use to search for files. fileMask can contain standard wildcard characters (* and ?). fileMask can contain full or partial file path.

Return: Returns comma-separated list of file names matching the specified mask.

Usage: On DOS/Windows based FTP hosts the FTP.dir statement is equivalent to DOS dir command. For most UNIX flavors, the FTP.dir statement is equivalent to UNIX ls command.

Note: If you don't include file path to the fileMask then FTP.dir statement returns files from the FTP server current directory.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
var list = FTP.dir(server, user, pass, "/pub/docs/*.html");
Scheduler.messageBox(list);
```

dirCreate

Prototype: FTP.dirCreate(String server, String user, String password, String dir)

Description: Creates a new directory on remote FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

dir - A string whose value is the full name of the directory to be created

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.dirCreate(server, user, pass, "/pub/dir1/dir2");
```

dirDelete

Prototype: FTP.dirDelete(String server, String user, String password, String dir)

Description: Deletes an existing directory on remote FTP server. If the directory is not empty all contained files or subdirectories are deleted recursively.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

dir - A string whose value is the full name of the directory to be deleted

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.dirDelete(server, user, pass, "/pub/dir1/dir2");
```

command

Prototype: FTP.command(String server, String user, String password, String command)

Description: Executes arbitrary commands directly on the specified FTP server.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

command - A string whose value is the command that you want to execute on the server

Usage: The FTP.command statement allows you to execute commands directly on the FTP server. The available commands vary depending on the type of server, and can usually be determined by logging on to the server with the command line FTP client and using the "remotehelp" command. A typical output of "remotehelp" command looks like the following:

```
ftp> remotehelp
  The following commands are recognized (* =>'s unimplemented).

   USER    PORT    STOR     MSAM*   RNTO    NLST     MKD     CDUP
   PASS    PASV    APPE     MRSQ*   ABOR    SITE     XMKD    XCUP
   ACCT*   TYPE    MLFL*    MRCP*   DELE    SYST     RMD     STOU
   SMNT*   STRU    MAIL*    ALLO    CWD     STAT     XRMD    SIZE
   REIN*   MODE    MSND*    REST    XCWD    HELP     PWD     MDTM
   QUIT    RETR    MSOM*    RNFR    LIST    NOOP     XPWD
```

 **Note:** You can use SITE EXEC command to execute operation system commands and run batch files and other programs on the FTP server computer. In order for the host command to be successfully executed your FTP server must support SITE EXEC command and this feature must not be disabled.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.command(server, user, pass,
       "SITE EXEC touch –mct 200506161400.05 test.txt");
```

config

Prototype: FTP.config(String property, String newValue)

Description: Set various parameters for subsequent FTP operations executed in the same job.

Parameters:

property - A string whose value is the name of the property that you want to change. The following properties are supported:

- "FTP PROTOCOL"
- "TRANSFER MODE"
- "LIST SEPARATOR"
- "PORT"
- "CONNECTION TYPE"
- "PRESERVE FILE TIMES"
- "SET TIME COMMAND"
- "TIME FORMAT"
- "TIME OFFSET"

new_value - A string whose value is the new value for the property that you want to change.

FTP PROTOCOL

The following values are supported for the "FTP PROTOCOL " property:

"FTP" – this is used for the classic FTP protocol

"SFTP" – this is used for secure FTP protocol which is an extension to SSH protocol.

"FTPS" – this is used for FTP over SSL channel protocol.

"SECURE" – this is a synonym for "SFTP" used for compatibility with the protocol name and options supported by 24x7 Scheduler Windows Edition.

"SSL" – this is a synonym for "FTPS" protocol name.

The default value is "FTP".

Use this property, to specify whether your want to use secure or non-secure FTP protocol for all subsequent FTP commands executed in the same job. This setting applies to all statements that belong to the FTP group.

TRANSFER MODE

The following values are supported for the "TRANSFER MODE" property:

- "ASCII"
- "BINARY"

The default value is "BINARY".

LIST SEPARATOR

For the "LIST SEPARATOR" property, you can specify any desired symbol that you will use to separate multiple files when performing multi-file FTP operations. The default value is comma. For more information, see FTP.getFile, FTP.resumeFile, FTP.appendFile, FTP.putFile, and FTP.deleteFile statements.

PORT

Use the "PORT" property, to specify which port your want to use for all subsequent FTP commands executed in the same job. This setting applies to all statements that begin with FTP prefix and also applies to the FTP.syncDir and FTP.compareDir statements. The default FTP port is 21.

CONNECTION TYPE

The following values are supported for the "CONNECTION TYPE " property:

- "PASSIVE"
- "ACTIVE"

Use this property, to specify whether your want to use active or passive FTP connection mode. This setting applies to all statements that begin with FTP prefix and to the FTP.syncDir and FTP.compareDir statements. If you do not change this property, the "ACTIVE" mode is used by default.

PRESERVE FILE TIMES
SET TIME COMMAND
TIME FORMAT
TIME OFFSET

Properties "PRESERVE FILE TIMES", "SET TIME COMMAND", "TIME FORMAT", "TIME OFFSET" are used together as a group. They control when and how to set date/time of transferred files.

 **Important Note:** Not all FTP servers support time commands. Check your FTP server documentation before attempting to use time-related properties.

The following values are supported for the "PRESERVE FILE TIMES" property:

- "TRUE"
- "FALSE"

Use this property, to specify whether your want to preserve times of uploaded and downloaded files. The default value is "FALSE" meaning that by default times of all uploaded and downloaded files are set to the current system time.

The following values are supported for the "SET TIME COMMAND" property:

- "" (an empty string)
- "[user specified host Operation System command]"

The default value is "" which instructs the script engine to use the extended version of the MDTM command supported by most modern FTP servers. If you specify some other non-empty value for the "SET TIME COMMAND" property the script engine will attempt to execute that command as a FTP host operation system command. For this purpose it executes FTP SITE EXEC command following by the specified host command. In order for the host command to be successfully executed your FTP server must support SITE EXEC command and this feature must not be disabled.

The "TIME FORMAT" property controls time format used with the user-defined command specified in the "SET TIME COMMAND" property. The default value for "TIME FORMAT" property is YYYYMMDDHHMM.SS.

The "TIME OFFSET" property can be used to specify the difference in time between the host and the target computer. In other words if the processing computer and the FTP server computer run in different time zones you can use this property to specify the time difference. Generally speaking you can use this property to affect how the file time is set when the value of "PRESERVE FILE TIMES" property is set to TRUE. Specify offset value in seconds. The default value of the "TIME OFFSET" property is 0. You can specify both positive and

negative values. A positive value causes the target file time to be adjusted forward; a negative value causes the target file time to be adjusted backward.

Example:

```
// Connect to the ftp server on port 25 with passive ftp and time
// diff -3 hours, and sync some directories.

FTP.config("PORT", 25);
FTP.config("CONNECTION TYPE", "PASSIVE");
FTP.config("PRESERVE FILE TIMES", "TRUE");
FTP.config("TIME OFFSET", "-10800");
FTP.config("TRANSFER MODE", "ASCII");
FTP.syncDir("LOCAL", "my.site.com", "testuser", "1111",
       "c:\\var\\www", "/pub/www/html/",
       true, false, true, true  );
```

compareDir

Prototype: CompareInfo FTP.compareDir(String server, String user, String password, String localDir, String remoteDir, boolean nameComparison)

Description: Compares files in two directories residing on local and remote computers using FTP protocol.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

local_dir - A string whose value is the name of the local directory containing files that you want to compare.

remote_dir - A string whose value is the name of the remote directory on FTP server containing files that you want to compare.

name_comparison - A boolean whose value should be TRUE if you want to compare files by name only, and FALSE if you want to compare them by name and date of the last modification.

Return: Returns CompareInfo object with the following functions:

String getLocalList () – Returns list of files from the local_dir directory, which are different from files in the remote_dir directory or could not be found in the remote directory.

String getRemoteList() – Returns list of files from the remote_dir directory, which are different from files in the local_dir directory or could not be found in the remote directory.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.config("PRESERVE FILE TIMES", "TRUE");
FTP.config("TIME OFFSET", "-10800");
var result = FTP.compareDir(server, user, pass, "e:\\Interpub\\wwwroot",
                            "/", false);
```

```
Scheduler.messageBox("Local files that are different remote files " +
       result.getLocalList());
Scheduler.messageBox("Remote files that are different from local files" +
       result.getRemoteList());
```

syncDir

Prototype: FTP.syncDir(String master, String server, String user, String password, String sourceDir, String targetDir, boolean addNew, boolean deleteMissing, boolean updateOld, boolean subDir)

Description: Synchronizes and replicates files across two directories residing on local and remote computers using FTP protocol.

Parameters:

server - A string whose value is FTP server host name (for example, ftp.microsoft.com) or IP address in ASCII dotted-decimal format (for example, 11.0.1.45)

user - A string whose value is the name of the user to log on to the server

password - A string whose value is the password to use to log on to the server

master - A string whose value instructs 24x7 Scheduler which computer is the "master" computer containing files and subdirectories that you want to replicate. The following values are supported:

- "REMOTE" - remote FTP server computer contains the "master" directory
- "LOCAL" - the local computer contains the "master" directory

source_dir - A string whose value is the name of the "master" directory containing files and subdirectories that you want to replicate

target_dir - A string whose value is the name of the target directory to which files and subdirectories are replicated

add_new - A boolean whose value should be TRUE if you want to replicate files that exist only in the source_dir, and FALSE otherwise

delete_missing - A boolean whose value should be TRUE if you want to delete from the target_dir directory these files that exist in the target_dir but do not exist in the source_dir, and FALSE otherwise

update_old - A boolean whose value should be TRUE if you want to update older versions of files in the target_dir directory, and FALSE otherwise. A file is considered as old if it exists in both target_dir and source_dir directories and the target_dir version of that file has a date time older than the version from the source_dir directory.

subdir - A boolean whose value should be TRUE if you want to update recursive subdirectories. Note that if you enable recursion then all other replication options described above apply to all subdirectories of all nesting levels starting with the source_dir.

Usage: Use FTP.syncDir statement to automate synchronization and replication for a group of files residing on different computers. The directory you are copying files from is also known as the master directory or primary site-replication

 **Note:** To update only old files set update_old parameter to TRUE and set both add_new and delete_missing parameters to FALSE.

To perform full 2-way file synchronization between 2 directories: run FTP.syncFTPDir using "REMOTE" for the master with delete_missing set to FALSE and everything else set to TRUE. Repeat FTP.syncFTPDir using "LOCAL" for the master with delete_missing set to FALSE and everything else set to TRUE.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
FTP.config("PRESERVE FILE TIMES", "TRUE");
FTP.config("TIME OFFSET", "-10800");
FTP.syncDir("LOCAL", server, user, pass, "c:\\buf2", "/pub",
       true,  // add new files
       false, // don't delete missing
       true,  // update old
       true   // recursively sync subdirectories
);
FTP.syncDir("REMOTE", server, user, pass, "/pub", "c:\\buf2",
       true,  // add new files
       false, // don't delete missing
       true,  // update old
       true   // recursively sync subdirectories
);
```

Mail Operations

send

Prototype: void Mail.send(String sender, String password, String recipient, String subject, String message)

Description: Establishes a new mail session and sends the specified mail message. The message is sent using the email server specified in scheduler's settings.

Parameters:

sender - A string whose is the sender's email address

password - A string whose value is the user's mail system password

recipient - A string variable whose value is the email address of the recipient for the message.

Note: To send the same message to multiple recipients you can specify their addresses as a comma-separated list

subject - A string variable whose value is the subject line, displayed in the message header

message - A string variable whose value is the content of the message body

Example:

```
Mail.send("my@mycompany.com", "password", "operations@mycompany.com",
       "test subject", "test message" );
```

sendWithAttachment

Prototype: void Mail.sendWithAttachment(String sender, String password, String recipient, String subject, String message, String attachments)

Description: Establishes a new mail session and sends the specified mail message. The message is sent using the email server specified in scheduler's settings.

Parameters:

sender - A string whose is the sender's email address

password - A string whose value is the user's mail system password

recipient - A string variable whose value is the email address of the recipient for the message.

 **Note:** To send the same message to multiple recipients you can specify their addresses as a comma-separated list

subject - A string variable whose value is the subject line, displayed in the message header

message - A string variable whose value is the content of the message body

attachment - A string variable whose value is the name of the file(s) to attach to the message.

 **Note:** To send multiple file attachments you can specify file names as a comma-separated list

Example:

```
Mail.sendWithAttachment("my@mycompany.com", "password",
       "operations@mycompany.com", "test subject", "test message",
       "/home/dir1/file1,/home/dir2/file2" );
```

Web Operations

getFile

Prototype: void Web.getFile(String url, String localFile)

Description: Downloads file from the specified URL. Downloaded file can be an HTML file or a file of any other type, including binary files.

Parameters:

url - A string whose value is the URL (Internet standard Uniform Resource Locator, e.g. Web address) that points to the web file that you want to download.

localFile - A string whose value is the name of a local file in which you want to save the returned data.

Examples:

```
Web.getFile("http://www.mycompany.com/hello.htm", "/home/myfiles/hello.htm" );
Web.getFile("http://www.mycompany.com/data/data.xls ",
        "C:\\ExcelFiles\\new_data.xls" );
```

postData

Prototype: void Web.postData(String url, String data, String localFile)

Description: Performs an HTTP POST, allowing a job to send a request through CGI, NSAPI, or ISAPI.

Parameters:

url - A string whose value is the URL (Internet standard Uniform Resource Locator, e.g. Web address) to post data to.

localFile - A string whose value is the name of the local file in which you want to save the web server response received after POST.

Examples:

```
Web.postData("http://www.mycompany.com/cgi-bin/add_customer.cgi",
       "name=Greg+Smith&company=ACME+Corp&phone=212-123-1234&" +
       "fax=212-123-5678&email=gsmith%40acme.com",
       "c:\\temp\\confirm.htm" );
```

Scheduler Operations

messageBox

Prototype: Scheduler.messageBox(String text)

Description: Displays graphical interactive message box containing user-defined message.

 **Important Notes:** Because the messageBox requires user intervention you should use this statement for job debugging purposes only and comment it out in production jobs. Also note that messageBox cannot be used in detached jobs and when 24x7 Scheduler is run in command console or run as a background daemon or Windows service.

Parameters:

text – text to be displayed.

Example:

```
Scheduler.messageBox("test message");
```

pause

Prototype: Scheduler.pause(int seconds)

Description: Causes the job to enter an efficient wait state until the specified time elapses.

Parameters:

seconds – pause duration in seconds.

Example:

```
Scheduler.pause(30);
```

logAddMessage

Prototype: void Scheduler.logAddMessage(String type, int jobId, String jobName, String message)

Description: Adds new message to the job log. The message can be viewed using the Log Viewer utility. If the job id matches an existing job the message also appears in the filtered job log.

Parameters:

type – A string constant indicating message type. Must be one of the following:

- "INFO"
- "WARNING"
- "ERROR"

jobId - ID of an existing job. If you don't want to hard-code job id of the active job you can use @V"job_id" macro in place of this parameter. Use zero for generic messages, not associated with any job.

jobName' - name of an existing job. If you don't want to hard-code job name of the active job you can use @V"job_name" macro in place of this parameter.

message – the message you want to add to the log.

Example:

```
Scheduler.logAddMessage('WARNING', 10, 'Test job',
       'No files found for processing. The job will abort now');
```

runJob

Prototype: int Scheduler.runJob(int jobID)

Description: Immediately runs the specified job and waits for the job to complete.

Parameters:

jobID - ID of an existing job.

Return: Returns unique run-time job number of the launched job.

Example:

```
var runId = Scheduler.runJob(125);
```

runRemoteJob

Prototype: int Scheduler.runRemoteJob(int jobID, String agent)

Description: Immediately runs the specified job and waits for the job to complete. The job execution place is controlled by the agent parameter.

Parameters:

jobID - ID of an existing job.

agent – Name of an agent profile configured in the scheduler settings for running remote jobs.

Return: Returns unique run-time job number of the launched job.

Example:

```
var runId = Scheduler.runRemoteJob(125, "QA server");
```

queueJob

Prototype: int Scheduler.queueJob(int jobID)

Description: Submits the specified job to the associated job queue according to the job priority. All currently running jobs continue running and not affected by the submitted job. The queue begins running the submitted job as soon as it completes running all jobs previously queued in the same queue with the same or higher priority and already running.

Parameters:

jobID - ID of an existing job.

Return: Returns unique run-time job number of the queued job.

Example:

```
var runId = Scheduler.queueJob(125);
```

queueRemoteJob

Prototype: int Scheduler.queueJob(int jobID, String agent)

Description: Submits the specified job to the associated job queue according to the job priority. All currently running jobs continue running and not affected by the submitted job. The queue begins running the submitted job as soon as it completes running all jobs previously queued in the same queue with the same or higher priority and already running.

Parameters:

jobID - ID of an existing job.

agent – Name of an agent profile configured in the scheduler settings for running remote jobs.

Return: Returns unique run-time job number of the queued job.

Example:

```
var runId = Scheduler.queueJob(125);
```

killJob

Prototype: Scheduler.killJob(int jobRunID)

Description: Terminates the specified job and in case of a program type job also removes the associated child processes. If the job is still queued and not yet started, killJob simply removes the job from the queue.

Parameters:

jobRunID - Run-time number of the job returned by Scheduler.runJob, Scheduler.queueJob, runRemoteJob statements.

Example:

```
var runID = Scheduler.queueJob(125);
Scheduler.pause( 60 );
Scheduler.killJob( runID );
```

deleteJob

Prototype: Scheduler.deleteJob(int jobID)

Description: Deletes the specified job from the job database. Any already queued or running instances of that job will remain in the job queue.

Parameters:

jobID - ID of an existing job.

Example:

```
Scheduler.deleteJob( 125 );
```

createJob

Prototype: int Scheduler.createJob(String JDL)

Description: Creates a new job and saves its definition in the job database.

Parameters:

JDL - The Job definition in JDL format. For a wide variety of examples see job templates available in the [24x7 install directory]\Template subdirectory. For a list and description of supported JDL commands see Job Properties in JDL Format topic.

Usage: Use Scheduler.createJob in your scripts to programmatically create new jobs. Use Scheduler.setJobProperty method to modify properties of existing job.

Return: Returns unique number identifying the created job in the job database.

Example:

```
var JDL =     "NAME=My job\n" +
              "JOB_TYPE=P\n" +
              "COMMAND=/bin/sh –c /home/john/mybatch.sh\n" +
              "SCHEDULE_TYPE=O\n" +
              "START_TIME=12:00\n" +
              "START_DATE=2005-10-17"
var jobID = Scheduler.createJob( JDL );
```

disableJob

Prototype: Scheduler.disableJob(int jobID)

Description: Disables the specified job in the job database. This will prevent the job from starting again. Any already queued or running instances of that job will remain in the job queue. As opposite to Scheduler.deleteJob, this statement does not delete the job definition.

Parameters:

jobID - ID of an existing job.

Example:

```
Scheduler.disableJob( 125 );
```

enableJob

Prototype: Scheduler.enableJob(int jobID)

Description: Enabled the specified job in the job database.

Parameters:

jobID - ID of an existing job

Example:

```
Scheduler.enableJob( 125 );
```

setJobProperty

Prototype: Scheduler.setJobProperty(int jobID, String propertyName, String newValue)

Description: Changes value of the specified job property in the job database. The change does not affect already queued or running instances of that job.

Parameters:

jobID - ID of an existing job

propertyName - The name of job property whose value you want to change For a list and description of supported JDL commands see Job Properties in JDL Format topic.

newValue - The new value for the specified property

Example:

```
Scheduler.setJobProperty( 125, "QUEUE", "Reports" );
Scheduler.setJobProperty( 125, "COMMAND", "/bin/sh –c /batch/report2.sh" );
```

getJobProperty

Prototype: String Scheduler.getJobProperty(int jobID, String propertyName)

Description: Returns value of the specified job property.

Parameters:

jobID - ID of an existing job

propertyName - The name of job property whose value you want to obtain. For a list and description of supported JDL commands see Job Properties in JDL Format topic.

Return: Returns current value of the requested job property

Example:

```
var command = Scheduler.getJobProperty( 125, "COMMAND" );
```

findJob

Prototype: Scheduler. int Scheduler.findJob(String name)

Description: Finds a job with the specified name.

Parameters:

name - A string whose value is the job name to search.

Return: Returns unique identifier for the found job or -1 if no match found. In case multiple jobs match the specified name, id of the first matching job is returned.

Example:

```
var jobId = Scheduler.findJob("File Mover" );
```

```
if (jobId != -1)
       Scheduler.runJob(jobId);
else
       Scheduler.raiseError(1001, "Unable to find File Mover job!");
```

getJobs

Prototype: int[] Scheduler.getJobs( )

Description: Returns array of job ids in the current job database.

Parameters:

none

Return: Returns unique identifiers for all jobs as an array of integer values.

Example:

```
// retrieve job command lines and disable all jobs
var jobArray = Scheduler.getJobs( );
for( var j = 0; j < jobArray.length; j ++)
{
       var command = Scheduler.getJobProperty(jobArray[j], "COMMAND" );
       Scheduler.disableJob( jobArray[j] );
}
```

getFolders

Prototype: int[] Scheduler.getFolders ( )

Description: Returns array of folder ids in the current job database.

Parameters:

none

Return: Returns unique identifiers for all folders as an array of integer values.

Example:

```
// retrieve array of folder identifiers
var folderArray = Scheduler.getFolders( );
```

raiseError

Prototype: Scheduler.raiseError(int errorCode, String errorMessage)

Description: Makes the job script to fail with the specified error code and message

Parameters:

erroCode - Numeric code value associated with the error

errorMessage – Text of the error message.

Example:

```
var server = "my server";
var user = "test";
var pass = "1111";
if (!FTP.fileExists(server, user, pass, "/pub/README"))
       Scheduler.raiseError(1001, "README file not found!");
```

stdError

Prototype: void Scheduler.stdError(String text)

Description: Prints message to the standard error stream, typically the console.

Parameters:

text - Text to output.

Return: None

Example:

```
Scheduler.stdError("Unable to delete file xyz.txt");
```

stdOutput

Prototype: void Scheduler.stdOutput(String text)

Description: Prints message to the standard output stream, typically the console.

Parameters:

text - Text to output.

Return: None

Example:

```
Scheduler.stdOutput("Successfully deleted file xyz.txt");
```

stdInput

Prototype: String Scheduler.stdInput()

Description: Reads text from the standard input stream, typically from the console.

Parameters:

None

Return: Returns text line from the input stream. WARNING: If the input stream is empty, the process will pause and wait for the input data.

Example:

```
var input = Scheduler.stdInput();
Scheduler.messageBox("You entered: " + input);
```

exitProcess

Prototype: void Scheduler.exitProcess(int exitCode)

Description: Terminates job process. Can be called in detached jobs only.

Parameters:

exitCode - Exit code of the terminated process.

Return: None

Example:

```
// terminate the current process with exit code 5
Scheduler. exitProcess(5);
```

Job Properties in JDL Format

All job properties are documented in the 24x7 Scheduler User's Guide. This topic can be used as a quick reference for supported job properties and their JDL names.

Job Definition Language (JDL) supports the following property names:

Property Name	Meaning
ACCOUNT	E-mail Account such as user ID, profile, or e-mail address (e-mail watch job). The actual value may differ for different e-mail interfaces. For a MAPI interface you should use the name of the MAPI profile you use when logging on to the e-mail system. For Lotus Notes you should use the name of the user (or ID) you use when logging on to the Lotus Notes. For SMTP you should use your e-mail address.
ALL_DAY_TYPE	All Day Schedule Type, one of the following: **R, L** (**R** - recursive, repeat at specified intervals; **L** - fixed time list)
AGENT	Same as Host (see Host description)
ASYNC	Asynchronous Process, one of the following: **Y**, **N** (yes, no)
BACKUP_AGENT	Same as Backup Host (see Backup Host description)
BACKUP_HOST	Backup Remote Host (e.g. Backup Remote Agent name)
CALENDAR	Name of the Calendar object assigned to the job
COMMAND	Program Command Line
DAY_END_TIME	Daily End Time for "all day" jobs with limited run-time interval
DAY_LIST	Monthly Schedule List of fixed Day Numbers, numbers must be in 1..31 range. Example: 1,3,5,7,14. This property is shared with TIME_LIST property for All Day Schedule.
DAY_NAME	Monthly Schedule Day Name, one of the following: **Monday** , **Tuesday** , **Wednesday** , **Thursday** , **Friday** , **Saturday** , **Sunday** , **Weekend** , **Weekday**
DAY_NUMBER	Monthly Schedule Day Number, a number from 1 – 31 range
DAY_START_TIME	Daily Start Time for "all day" jobs with limited run-time interval
DELAY	Allowed Job Delay Interval (minutes)
DELETE_RULE	Delete, Move, and Rename Semaphore File Rules, one of the following: **D**, **A**, **B, M, E, R, C, F, G** (**D** - do not delete, move, rename; **A** - delete after job run; **B** - delete before job run; **M** - move before job run; **E** - move after job run; **R** - rename before job run; **C** - rename after job run; **F** - move and rename before job run; **G** - move and rename after job run)
DESCRIPTION	Job Description
DISABLE_ON_ERROR	Disable Job on Error, one of the following: **Y**, **N** (yes, no)
DISABLED	Job Disabled Status, one of the following: **Y**, **N** (yes, no)
DETACHED	Detached Job, one of the following: **Y**, **N** (yes, no).  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.

END_DATE	Last Job Start Date
END_TIME	Last Job Start Date
EXIT_CODE	Exit Code Condition (as a string expression)
FILE	Semaphore File Names(s) for file-watch jobs; Module Name for process-watch job
FOLDER	Job Folder ID. This is read-only property. It may not be changed using **SET** command. It can be retrieved using **GET** command
FOLDER_NAME	Job Folder Name. This is read-only property. It may not be changed using **SET** command. It can be retrieved using **GET** command
FRIDAY	Execute Job On Fridays, one of the following: **Y**, **N** (yes, no)
HOST	Remote Host (Remote Agent Name)
ID	Job ID, This is read-only property. It may not be changed using **SET** command. It can be retrieved using **GET** command with Job Name parameter.
IGNORE_ERRORS	Ignore Errors, one of the following: **Y**, **N** (yes, no)
INIT_TIMEOUT	Initial Timeout before sending keystroke (seconds)
INTERVAL	Repeat Interval for Job having Schedule Type **T**
JOB_PASSWORD	Job Protection State and Password. Sets or removes job protection state and password. This is a write-only property. It can be changed using **SET** command, but it cannot be retrieved using **GET** command. The value in this property must be specified in the following format : [old password][tab character][new password][tab character][protection state] If the job is not protected, the [old password] is ignored, otherwise a valid password must be specified in order to remove or change job password or protection state. If the protection exists and the new protection state is specified as an empty string the protection will be removed. The protection code must one of the following: **F**, **E, R,** an empty string (**F** – full protection; **E** – execute only; **R** – read only; an empty string indicates that a job is not protected).
JOB_TYPE	Job Type, one of the following: **P**, **D**, **S** (program, database, script)
KEYSTROKE	Keystroke to send to the launched program.  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
LOG	Log Job Execution, one of the following: **Y**, **N** (yes, no)
MESSAGE	E-mail Message Text (e-mail watch job)
MODIFY_TERMINAL	Network name of the computer from which the job was last modified. This is a read-only property. It cannot be changed using **SET** command. It can be retrieved using **GET** command.  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MODIFY_TIME	Date and time when the job was modified. This is a read-only property. It cannot be changed using **SET** command. It can be retrieved using **GET** command.  **Note:** This property is not used in 24x7 Scheduler Multi-platform

	Edition.
MODIFY_USER	Name of the user who last modified the job. This is a read-only property. It cannot be changed using **SET** command. It can be retrieved using **GET** command.  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MONDAY	Execute Job On Mondays, one of the following: **Y**, **N** (yes, no)
MONTHLY_TYPE	Monthly Schedule Type, one of the following: **D**, **T, L** (**D** - by day number; **T** - by day name; **L** - fixed day list)
MOVE_DIR	Name of the destination directory for semaphore file move and rename operations.
MSG_ACCOUNT	E-mail Account for Notification Action of E-mail Type E-mail (user ID, profile, or e-mail address). The actual value may differ for different e-mail interfaces. For the MAPI interface you should use the name of the MAPI profile you use when logging on to the e-mail system. For Lotus Notes you should use the name of the user (or ID) you use when logging on to Lotus Notes. For SMTP you should use your e-mail address.
MSG_ACTIONS	Map of Notification Actions and Events in text format. The map is represented as a comma-separated list of 2-character values where in every list item the first character represents Notification Event Type and the second character represents Notification Action Type. The following characters can be used for the event type: **S** – job start, **F** – job finish, **E** – job error, **N** – job file not found, **L** – job is late. The following characters can be used for the action type: **E** – send email, **P** – send page, **N** – send network popup message, **D** – execute database commands, **F** – create semaphore files, **T** – send SNMP trap, **J** – run job, **R** – run program, **S** – run script. Example map: SE,FE,EE,FD This example map represents the following Notification Events and Events: 1. Send notification email on job start. 2. Send notification email on job finish. 3. Send notification email on job error. 4. Execute database commands on job finish.
MSG_DATABASE	Execute Notification Action of Database Type, one of the following: **Y**, **N** (yes, no)
MSG_E-MAIL	Execute Notification Action of E-mail Type, one of the following: **Y**, **N** (yes, no)
MSG_ERROR	Execute Notification Action on Job Execution Error, one of the following: **Y**, **N** (yes, no)
MSG_FILE	Execute Notification Action of Semaphore File Type, one of the following: **Y**, **N** (yes, no)
MSG_FILE_NAME	File name(s) for Notification Action of Semaphore File Type
MSG_FINISH	Execute Notification Action on Job Finish, one of the following: **Y**, **N** (yes, no)
MSG_JOB	Execute Notification Action of Run Job Type, one of the following: **Y**, **N** (yes, no)

MSG_JOB_ID	Job name or job id for Notification Action of Run Job Type
MSG_LATE	Execute Notification Action on Job Late Start, one of the following: **Y**, **N** (yes, no)
MSG_NET	Execute Notification Action of Network Message Type, one of the following: **Y**, **N** (yes, no)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_NET_RECIPIENT	Message Recipient for Notification Action of Network Message Type  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_NOTFOUND	Execute Notification Action on Job Executable Not Found Error, one of the following: **Y**, **N** (yes, no)
MSG_PAGE	Execute Notification Action of Page Type, one of the following: **Y**, **N** (yes, no)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_PAGER	Page Recipient's Pager Number  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_PASSWORD	E-mail Password for Notification Action of E-mail Type
MSG_PROFILE	Database Profile for Notification Action of Database Type
MSG_PROGRAM	Execute Notification Action of Run Program Type, one of the following: **Y**, **N** (yes, no)
MSG_PROGRAM_NAME	Program name for Notification Action of Run Program Type
MSG_PROGRAM_TIMEOUT	Process Timeout (seconds) for Notification Action of Run Program Type
MSG_RECIPIENT	E-mail Recipient for Notification Action of E-mail Type
MSG_SCRIPT	Script for Notification Action of Script Type  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_SCRIPT_TYPE	Type of Script for Notification Action of Script Type, one of the following: **JAL**, **VBS** (Job Automation Language, Visual Basic Script)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
MSG_START	Execute Notification Action on Job Start, one of the following: **Y**, **N** (yes, no)
MSG_TRAP	Execute Notification Action of Send SNMP Trap Type, one of the following: **Y**, **N** (yes, no)
MULTI_INSTANCE_CONTROL	Job queuing rule for handling multiple job instances , one of the following: R – always queue and run, **T** – terminate already running and queued job instances if any, and add new instance, **S** – if there are already running or queued job instances, skip new instance and do nothing, **E** – if there are already running or queued job instances, skip new instance and raise an error.
NAME	Job Name

NUMBER_OF_RETRIES	Maximum Number of Retries Before Job Gets Marked as Failed (number)
PASSWORD	E-mail Password (e-mail watch job)
POLLING_INTERVAL	Polling Interval (minutes)
PRIORITY	Job Priority, one of the following: **-1** – low, **0** – normal, **1** – high
PROFILE	Database Profile
PROTECTION	Job Protection State, one of the following: **F**, **E, R,** an empty string (**F** – full protection; **E** – execute only; **R** – read only; an empty string indicates that a job is not protected). This is a read-only property. It cannot be changed using **SET** command. It can be retrieved using **GET** command.
QUEUE	Job Queue
REBOOT	Reboot Computer After Job Finished, one of the following: **Y**, **N** (yes, no)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
RETRY_INTERVAL	Retry Interval (seconds)
RENAME_SUFFIX	Retry Interval (seconds)
RETRY_ON_ERROR	The name suffix used in semaphore file names for rename operations.
RUNAS_DOMAIN	Domain Name for authentication of jobs to be run using another user account.  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition. In that version the RUNAS_USER must contain both the domain and user names in **domain\user** format.
RUNAS_PASSWORD	Password for authentication of jobs to be run using another user account.
RUNAS_USER	User Name for authentication of jobs to be run using another user account.
SATURDAY	Execute Job On Saturdays, one of the following: **Y**, **N** (yes, no)
SAVE_ATTACHMENT	Save E-mail Attachments (e-mail watch job), one of the following: **Y**, **N** (yes, no)
SCHEDULE_TYPE	Schedule Type, one of the following: **O**, **D**, **T**, **M**, **F**, **P**, **A**, **E, I, L, S** (Time trigger: **O** – run once, **D** – repeat daily, **T** – repeat at specified time interval, **M** – repeat monthly; File trigger: **F** – check semaphore files; Process trigger: **P** – check process presence, **A** – check process absence; E-mail trigger: **E** - check e-mail message; User trigger: **I** – wake up on "user idle" event, **L**- wake up on log-off event, **S** – wake up on shutdown event)
SCRIPT	Job Script
SCRIPT_TYPE	Job Script Type, one of the following: **JAL**, **VBS, JS** (Job Automation Language, Visual Basic Script, JavaScript)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
SEND_KEYSTROKE	Send Keystroke, one of the following: **Y**, **N** (yes, no)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.
SIZE_CHECK_INTERVAL	File Size Stability Check Interval (used in File-watch jobs)
SKIP	Skip Late Job, one of the following: **Y**, **N** (yes, no)

SKIP_HOLIDAY	Skip Job on Holiday, one of the following: **Y**, **N** (yes, no)
SLIDE_HOLIDAY	Slide Job Execution Time on the next non-holiday if it falls on holiday, one of the following: **Y**, **N** (yes, no)
SQL	SQL Command(s)
START_DATE	First Start Date
START_IN	Program Start-up Directory
START_TIME	First Start Time
SUBJECT	E-mail Message Subject for (e-mail watch job)
SUNDAY	Execute Job On Sundays, one of the following: **Y**, **N** (yes, no)
TIME_LIST	All Day Schedule List of fixed Times, values must be in valid 24-hour time format. Example: 11:10,12:10,17:10,18:10. This property is shared with DAY_LIST property for Monthly Schedule.
THURSDAY	Execute Job On Thursdays, one of the following: **Y**, **N** (yes, no)
TIMEOUT	Timeout (seconds)
TUESDAY	Execute Job On Tuesdays, one of the following: **Y**, **N** (yes, no)
WEDNESDAY	Execute Job On Tuesdays, one of the following: **Y**, **N** (yes, no)
WINDOW	Window Style, one of the following: **N**, **M**, **I**, **H** (normal, maximized, iconic, hidden)  **Note:** This property is not used in 24x7 Scheduler Multi-platform Edition.

Additional Java Script Documentation and Examples

Online JavaScript documentation is available at these sites:

http://developer.netscape.com - Netscape's JavaScript Guide, Reference, and more

http://msdn.microsoft.com/workshop - Microsoft's JScript documentation

In addition, you can find lots of JavaScript discussions, online tutorials, links, code examples, and hundreds of thousands of useful scripts at many other sites. Some useful sites are

http://javascript-reference.info

http://www.javascripts.com

http://www.javascripter.net

www.regular-expressions.info/javascript.html

www.webreference.com/programming/javascript