



The Small Language Scripting Reference

RemotelyAnywhere uses a third party scripting language called Small.

WHAT IS SMALL?

This is probably best described in the words of the author of the language itself:

Small is a simple, typeless, 32-bit extension language with a C-like syntax. The Small compiler outputs P-code (or bytecode) that subsequently runs on an abstract machine. Execution speed, stability, simplicity and a small footprint were essential design criterions for both the language and the abstract machine.

[...]

The Small language was designed as a flexible, general-purpose language. The tool set (compiler, abstract machine) were written so that they were easily extensible and would run on different software/hardware architectures.

Many years ago, I retyped the “Small C” compiler from Dr. Dobb’s Journal, by Ron Cain and James Hendrix. Having just grasped the basics of the C language, working on the Small C compiler was a learning experience of its own. [...]

In early 1998, I was looking for a scripting language for an animation toolkit. [...] While experimenting with Quincy (from Al Stevens), I decided that a simplified C would probably be a good fit. I dusted off Small C. This is the result.

Small is a descendent of the original Small C, which at its turn was a subset of C. The most fundamental changes that I did were the removal of the type system and the substitution of pointers by references. The motivations to adapt the C language to (yet another) tiny language are best discussed elsewhere (see the rationale in appendix A), but by scrapping the type system and the support for pointers, I could hardly call my language a “subset of C” or a “C dialect”. Therefore, I stripped off the “C” from the title and kept the name “Small”.

Thiadmer Riemersma, the Small Booklet, 1999

For an introduction and a complete language reference, please see the Small booklet on the RemotelyAnywhere website: <http://www.remotelyanywhere.com/smalldoc.pdf>.

This document describes the RemotelyAnywhere extensions to the language. These extensions allow your scripts to communicate with RemotelyAnywhere, the user, and provide a small subset of the Win32 API.

The definition of the functions are in the ‘ra.inc’ header file, so you must insert the “#include <ra>” line in your script’s source code in order to use them. If you create a new script with RemotelyAnywhere, it will place the appropriate line in the source to use these functions.

The following is a list and description of functions that make up the extensions and the page on which they can be found in this document:

native gb(arr[], id);.....	6
native sb(arr[], id, value);.....	6
native strize(array[], len=-1);.....	6
native strlen(string[]);.....	7
native strcpy(dest[], src[]);.....	7
native strcat(dest[], src[]);.....	8
native strleft(dest[], src[], len);	8
native strright(dest[], src[], len);.....	9
native strmid(dest[], src[], pos, len);.....	9
native strchr(string[], c, start=0);.....	10
native strstr(string[], string2[], start=0);.....	10
native strcmp(string[], string2[]);	11
native stricmp(string[], string2[]);	11
native sprintf(string[], format[], ...);	11
native atoi(string[]);	12
native fopen(filename[], mode);.....	13
native fclose(file);.....	13
native fread(file, buf[], bytes);.....	13
native fwrite(file, buf[], bytes);.....	14
native fseek(file, offset, method);.....	14

native socket(address[], port, timeout=-1);.....	15
native closesocket(sock);.....	16
native recv(sock, buf[], bytes);	16
native send(sock, buf[], bytes);	17
native htmlBeginOutput(title[]="");	17
native htmlEndOutput();	18
native htmlWrite(text[], htmlize=false);	18
native htmlBR();	18
native htmlBeginTable(...);	18
native htmlEndTable();.....	19
native htmlTableRow(...);	19
native htmlBeginTableRow();.....	19
native htmlEndTableRow();.....	19
native htmlTableCell(text[]);.....	19
native htmlBeginTableCell();.....	19
native htmlEndTableCell();.....	19
native htmlBeginDialog(text[]);.....	19
native htmlEndDialog();	20
native htmlBeginForm();.....	20
native htmlEndForm();.....	20
native htmlAddParam(...);	20
native htmlGetParam(param[], value[]);.....	20
native htmlButton(text[], func[]);.....	21
native htmlButtonBack(text[]="Back", form=true);.....	21
native htmlRadioButton(param[], value[], checked=false);.....	22
native htmlCheckbox(param[], value[], checked=false);	22
native htmlEdit(param[], value[]="");	23
native htmlLink(title[], url[]);	23
native htmlCBLINK(title[], func[], ...);.....	23
native htmlError(msg[]);.....	24

native raEnumProcs();.....	24
native raGetProcessNum();	24
native raGetProcess(id, &pid, name[], &cpu, &mem);.....	24
native raGetNextProcess(&pid, name[], &cpu, &mem);.....	24
native raEnumProcsClose();.....	25
native raKillProcess(pid);.....	25
native raForkProcess(cmdline[]);.....	26
native raExecuteCmd(cmd[], buf[]="", buflen=0, timeout=-1);.....	26
native raReboot(type=REBOOT_NORMAL);.....	27
native raEnumServices();.....	27
native raGetServiceNum();	28
native raGetService(id, name[], displayName[], binary[], &type, &status, &startup);.....	28
native raGetNextService(name[], displayName[], binary[], &type, &status, &startup);	28
native raEnumServicesClose();.....	29
native raStartService(name[]);.....	29
native raStopService(name[]);.....	30
native raPauseService(name[]);.....	30
native raContinueService(name[]);.....	30
native raGetTime();	31
native raGetPerformance(type, time, &value, ...);.....	31
native raSleep(time);.....	32
native raLog(msg[]);.....	32
native raSendMail(to[], subj[], msg[], from[]="");.....	32
native raMessage(to[], msg[]);	33
native raRegGetValue(base, key[], value[], type, ...);.....	33
native raRegSetValue(base, key[], value[], type, ...);	35

native gb(arr[], id);

Extracts a byte from an array at the specified index. (Handles a Small cell array as a byte array.)

Parameters

arr

[in] The array to extract the byte from

id

[in] The index from the byte is to be extracted

Return Value

If the function succeeds, it returns the value of the byte at

the specified index in the array (in the range [0..255]).

If the function fails, it returns -1.

native sb(arr[], id, value);

Sets a byte in an array at the specified index. (Handles a Small cell array as a byte array.)

Parameters

arr

[in] The array to set the byte in

id

[in] The index at the byte is to be set

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns 0.

native str2cstr(array[], len=-1);

Converts a Small string to a C-Style string and vice versa. (It is useful when - for example - you want to read/write a text message from/to a file or a socket.)

Parameters

array

[in] The array containing the text to be converted.

len

[in] The length of the string to be converted. If this parameter is -1, the string is assumed to be zero terminated and all characters are going to be converted.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns 0.

native strlen(string[]);

Returns the length in characters of a string (not including the terminating null character).

Parameters

string

[in] The string whose length is to be returned

Return Value

If the function succeeds, it returns the length of the string (greater than or equals to 0).

If the function fails, it returns -1.

native strcpy(dest[], src[]);

Copies a string to a buffer.

Parameters

dest

[out] The array to receive the contents of the src string. The array must be large enough to contain the string including the terminating null character.

src

[in] The string to be copied.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns 0.

native strcat(dest[], src[]);

Appends one string to another.

Parameters

dest

[in/out] The array containing the string to which the contents of src are to be appended.

The array must be large enough to contain both strings.

src

[in] The string to be appended.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns 0.

native strleft(dest[], src[], len);

Extracts the beginning of a string to an array.

Parameters

dest

[out] The array to receive the contents of the src string's specified part. The array must

be large enough to contain it including the terminating null character.

src

[in] The string to extract from.

len

[in] The number of characters to be copied from the beginning of src to dest. If the

length of src is less than len, the whole string is being copied.

Return Value

If the function succeeds, it returns the number of characters copied to dest not including the terminating null character.

If the function fails, it returns 0.

native strlgt(dest[], src[], len);

Extracts the end of a string to an array.

Parameters

dest

[out] The array to receive the contents of the src string's specified part. The array must be large enough to contain it including the terminating null character.

src

[in] The string to extract from.

len

[in] The number of characters to be copied from the end of src to dest. If the length of src is less than len, the whole string is being copied.

Return Value

If the function succeeds, it returns the number of characters copied to dest not including the terminating null character.

If the function fails, it returns 0.

native strmid(dest[], src[], pos, len);

Extracts the specified part of a string to an array.

Parameters

dest

[out] The array to receive the contents of the src string's specified part. The array must be large enough to contain it including the terminating null character.

src

[in] The string to extract from.

pos

[in] The index of the first character that is to be copied from src.

len

[in] The number of characters to be copied from src to dest. If the length of src is less than pos+len, only the appropriate number of characters are being copied.

Return Value

If the function succeeds, it returns the number of characters copied to dest not including the terminating null character.

If the function fails, it returns 0.

native strchr(string[], c, start=0);

Finds the first occurrence of a character in a string.

Parameters

string

[in] The string to search in.

c

[in] The character to be searched.

start

[in] The index from which the search to start.

Return Value

If the character is found, the index of the first occurrence of c in string.

If the function fails or the character is not found, -1.

native strstr(string1[], string2[], start=0);

Finds the first occurrence of a string in another string.

Parameters

string1

[in] The string to search in.

string2

[in] The string to search for.

start

[in] The index which the search to start at.

Return Value

If the string is found, the index of the first occurrence of string2 in string1.

If the function fails or the string is not found, -1.

native strcmp(string1[], string2[]);

native stricmp(string1[], string2[]);

Compares (case sensitive/insensitive) two strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings.

The function returns the difference of the values of the first unequal characters it encounters. For example, strcmp determines that “abcz” is greater than “abcdefg” and returns the difference of z and d.

Parameters

string1

[in] The first string.

string2

[in] The second string.

Return Value

If string1 is less than string2, the return value is negative. If string1 is greater than string2, the

return value is positive. If the strings are equal, the return value is zero.

If the function fails, the return value is zero.

native sprintf(string[], format[], ...);

Formats a string. The parameters are converted and placed in the output string according to the format specifications in the format parameter.

Parameters

string

[out] The array that receives the formatted string. It must be large enough to contain the resulting string.

format

[in] The format specification string. For more information, see the Remarks section.

Return Value

If the function succeeds, the return value is the number of characters stored in the string array not including the terminating null character.

If the function fails, the return value is zero.

Remarks

The format specification is very similar to that of the sprintf function of the standard C library. Fields always begin with a percent sign (%). If an unrecognized character follows a percent sign, it is inserted into the output. A format specification has the following form:

`%[o][width]type`

The width specifies the minimum field width allocated to the parameter in the output string. The allocated field is always filled right aligned with the parameter. If the width is preceded by a 'o' character, and the parameter is an integer type, the blank space remaining in the field is filled with 'o' characters.

The following type specifications are supported:

Field	Meaning
c	Single character. The value is interpreted as the ASCII code of a character.
d	Signed decimal integer. This type is equivalent to i.
i	Signed decimal integer. This type is equivalent to d.
s	String. The value is interpreted as an array of characters.
t	Time. The value is interpreted as an integer returned by <code>raGetTime()</code> : the number of seconds elapsed since midnight, January 1, 1970. The output is formatted as "hh:mm:ss".
T	Date. The value is interpreted as in the case of the 't' type. The output is formatted as "DD-MM-YYYY".
U	Unsigned decimal integer.
x	Unsigned hexadecimal integer with lowercase alphabetical characters.
X	Same as 'x', but with uppercase alphabetical characters.

native atoi(string[]);

Converts a string to an integer.

Parameters

string

[in] The string to be converted.

Return Value

Returns the value produced by interpreting the input string as a signed decimal number. The return value is 0 if the string cannot be interpreted. The return value is undefined in the case of an overflow.

native fopen(filename[], mode);

Opens a file.

Parameters

filename

[in] A string representing the name of the file to be opened.

mode

[in] A number indicating how to open the file. It can have the following values:

FILE_READ: Open the file for reading

FILE_WRITE: Open the file for writing

Return Value

If the function succeeds, it returns a nonzero value identifying the opened file.

If the function fails, it returns zero.

native fclose(file);

Closes an open file.

Parameters

file

[in] A file identifier returned by fopen.

native fread(file, buf[], bytes);

Reads data from a file.

Parameters

file

[in] A file identifier returned by fopen.

buf

[in/out] An array that receives the file contents.

bytes

[in] Number of bytes to be read into the array.

Return Value

If the function succeeds, it returns the number of bytes read, which may be less than bytes if the end of file is reached or an error encountered.

If the function fails, it returns zero.

native fwrite(file, buf[], bytes);

Writes data to a file.

Parameters

file

[in] A file identifier returned by fopen.

buf

[in] An array whose contents are to be stored.

bytes

[in] Number of bytes to be stored.

Return Value

If the function succeeds, it returns the number of bytes written, which may be less than bytes if an error is encountered.

If the function fails, it returns zero.

native fseek(file, offset, method);

Repositions the file pointer.

Parameters

file

[in] A file identifier returned by fopen.

offset

[in] Offset, which the file pointer is to be set to.

method

[in] Identifies the initial position, which the file pointer is to be set relatively to. It must be one of the following constants:

FILE_BEGIN: beginning of the file
FILE_CURRENT: current file position
FILE_END: end of the file

Return Value

If the function succeeds, it returns the number of bytes written, which may be less than bytes if an error is encountered.

If the function fails, it returns zero.

native socket(address[], port, timeout=-1);

Opens a TCP socket and connects it to the specified address and port.

Parameters

address

[in] A string representing the address the socket is to be connected to. It may be a dotted number (for example “127.0.0.1”) or a machine name to be resolved (for example “www.remotelyanywhere.com”).

port

[in] The port number, which the socket is to be connected to.

timeout

[in] Timeout value in milliseconds to be assigned to the socket. If -1, no timeout is assigned.

Return Value

If the function succeeds, it returns a nonzero number identifying the socket.

If the function fails, it returns zero.

Remarks

RemotelyAnywhere currently supports blocking mode TCP sockets only. The specified timeout value will affect the behavior of all operations on the socket. If a request cannot be completed in the time interval specified by timeout, the calls will fail. If timeout is not specified, all operations will block until they can be finished or some error occurs.

native closesocket(sock);

Closes a socket.

Parameters

sock

[in] An identifier returned by socket.

native recv(sock, buf[], bytes);

Reads data from a socket.

Parameters

sock

[in] A socket identifier returned by socket.

buf

[in/out] An array that receives the data.

bytes

[in] Maximum number of bytes to be read into the array.

Return Value

If the function succeeds, it returns the number of bytes read.

If the function fails, it returns zero.

Remarks

The function will block until some input is available or an error occurs (for example the timeout specified in socket elapses).

native send(sock, buf[], bytes);

Sends data to a socket.

Parameters

sock

[in] A socket identifier returned by socket.

buf

[in] An array whose contents are to be sent.

bytes

[in] Number of bytes to be sent.

Return Value

If the function succeeds, it returns the number of bytes written.

If the function fails, it returns zero.

Remarks

The function will block until all the data are sent or some error occurs (for example the timeout specified in socket elapses).

native htmlBeginOutput(title[]="");

Opens the output to be sent to the caller of the script.

Parameters

title

[in] A string specifying the text that should appear in the headline of the

RemotelyAnywhere browser window.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns zero.

Remarks

The function will succeed if the script containing it is called from the RemotelyAnywhere user interface. It fails, if it is called from a Monitoring Script rule. If this function fails any further call to the html... functions should be skipped because they will have no effect at all.

The htmlEndOutput() function must be called to send the output.

native htmlEndOutput();

Sends the output to the caller of the script.

native htmlWrite(text[], htmlize=false);

Writes text to the HTML output.

Parameters

text

[in] The string to be written to the output.

htmlize

[in] Specifies whether text is to be converted to html format. See the Remarks section for more information.

Remarks

The htmlize parameter specifies the way the HTML output is being written. If it is false, you can directly write the output using HTML tags, special characters, etc. If it is true, the text is converted to appear the same in the browser window. Special characters (for example brackets, national characters, etc.) are substituted to their HTML equivalent.

native htmlBR();

Writes a line break tag (
) to the output.

native htmlBeginTable(...);

Begins a table on the output.

Parameters

...

[in] Strings that specify the name of the columns in the table

Remarks

htmlEndTable() should be called in order to the table appear correctly.

native htmlEndTable();

Ends a table started with htmlBeginTable.

native htmlTableRow(...);

Writes a row to a table.

Parameters

...

[in] Strings that specify the content of the cells in the row

native htmlBeginTableRow();

Begins a row in the current table.

native htmlEndTableRow();

Ends a row started with htmlBeginTableRow.

native htmlTableCell(text[]);

Inserts a cell into the current row of the table.

Parameters

text

[in] String specifying the content of the cell

native htmlBeginTableCell();

Begins a cell in the current row of the table.

native htmlEndTableCell();

Ends the current cell started with htmlBeginTableCell.

native htmlBeginDialog(text[]);

Begins a dialog area on the output. It will have the standard “RemotelyAnywhere look-and-feel”.

Parameters

text

[in] String specifying the caption of the dialog

native htmlEndDialog();

Ends a dialog started with htmlBeginDialog.

native htmlBeginForm();

Begins a form on the output.

native htmlEndForm();

Ends a form started with htmlBeginForm.

native htmlAddParam(...);

Inserts so-called hidden parameters into the current form (started with htmlBeginForm) that can be accessed with htmlGetParam when processing form data.

Parameters

...

[in] String pairs specifying the name and value of parameters

native htmlGetParam(param[], value[]);

Gets the value of a parameter submitted on a form or in a link.

Parameters

param

[in] String specifying the name of the parameter whose value is to be get

value

[out] Array that receives the string value of the specified parameter. It must be large

enough to contain the string.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns zero.

Remarks

This call will succeed only in a callback function specified in htmlButton or htmlCBLINK.

native htmlButton(text[], func[]);

Inserts a button into a form that submits the form data to the specified callback function.

Parameters

text

[in] String specifying the caption of the button.

func

[in] String specifying the name of the callback function to call when the button is clicked.

Return Value

If the function succeeds, it returns 1.

If the function fails, it returns zero.

Remarks

This function is valid only on a form started with htmlBeginForm.

native htmlButtonBack(text[]="Back", form=true);

Puts a button that will navigate the user back to the Scripts page of RemotelyAnywhere.

Parameters

text

[in] String specifying the caption of the button.

form

[in] A boolean value specifying whether to insert a new form containing this button only. See the Remarks section for more information.

Remarks

If you call this function in the scope of a form started with htmlBeginForm you should pass false in the form parameter, because it is of unpredictable result creating a form without closing the previous one.

native htmlRadioButton(param[], value[], checked=false);

Inserts a radio button into a form with the specified name and value.

Parameters

param

[in] String specifying the parameter, which the radio button will modify.

value

[in] String specifying the value, which the radio button will insert into the parameter.

This will also be the caption of the radio button.

checked

[in] Boolean value specifying whether the radio button is checked by default.

Remarks

This function is valid only in the scope of a form started with htmlBeginForm.

Radio buttons assigned to the same parameter belong to the same group. Only one of them is allowed to be checked, because it provides the value of the parameter.

native htmlCheckbox(param[], value[], checked=false);

Inserts a checkbox on a form with the specified name and value.

Parameters

param

[in] String specifying the parameter, which the checkbox will modify.

value

[in] String specifying the value, which the checkbox will insert into the parameter. This will also be the caption of the checkbox.

checked

[in] Boolean value specifying whether the checkbox is checked by default.

Remarks

This function is valid only in the scope of a form started with htmlBeginForm.

The form will submit the parameter only when the checkbox is checked.

native htmlEdit(param[], value[]="");

Inserts an edit line into a form.

Parameters

param

[in] String specifying the parameter, which receives the content of the edit line.

value

[in] String specifying the initial value of the edit line.

Remarks

This function is valid only in the scope of a form started with htmlBeginForm.

native htmlLink(title[], url[]);

Writes a link to the output.

Parameters

title

[in] String specifying the text appearing on the output.

url

[in] String containing the URL, which the link points to.

native htmlCBLLink(title[], func[], ...);

Writes a link that will call a function in the current script with the specified parameters.

Parameters

title

[in] String specifying the text appearing on the output.

func

[in] String specifying the name of the callback function to be called when the link is activated.

...

[in] String pairs containing parameter name-value information, that will be passed to the callback function.

Remarks

The value of parameters passed with the link can be retrieved with htmlGetParam in the callback function.

native htmlError(msg[]);

PUTS A STANDARD REMOTELYANYWHERE ERROR DIALOG TO THE OUPUT.

Parameters

msg

[in] String specifying the text appearing in the error box.

Remarks

This call will discard the previously written output and immediately sends the error message to the user, so no further output should be written.

native raEnumProcs();

TAKES A SNAPSHOT OF THE CURRENTLY ACTIVE PROCESSES, THAT CAN BE ENUMERATED.

Remarks

You can call raGetProcessNum, raGetProcess, raGetNextProcess to retrieve information on the processes. You must call raEnumProcsClose to release resources allocated by the snapshot.

native raGetProcessNum();

GETS THE NUMBER OF PROCESSES IN THE LAST SNAPSHOT TAKEN WITH RAENUMPROCS.

native raGetProcess(id, &pid, name[], &cpu, &mem);

native raGetNextProcess(&pid, name[], &cpu, &mem);

RETRIEVES INFORMATION ON THE SPECIFIED PROCESS IN THE LAST SNAPSHOT TAKEN WITH RAENUMPROCS.

Parameters

id

[in] The index of the process to retrieve information on.

pid

[out] Variable that receives the process ID.

name

[out] Array that is being filled with the string representing the name of the process.

cpu

[out] Variable that receives the CPU time the process used in milliseconds.

mem

[out] Variable that receives the size of memory allocated by the process in Kilobytes.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raEnumProcsClose();

Frees resources allocated by raEnumProcs.

native raKillProcess(pid);

Terminates the process with the specified ID.

Parameters

pid

[in] Process ID.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

Remarks

Windows might reuse the process ID of a process when it is terminated, so you must be very careful when getting the ID from a snapshot taken with raEnumProcs and be sure that the process you are going to terminate is still running. You should call raKillProcess as soon as possible after raEnumProcs.

native raForkProcess(cmdline[]);

Executes a command using the CreateProcess Win32 function.

Parameters

pid

[in] String specifying the command line.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

Remarks

The function will succeed if the executable specified in the command line is found regardless to if it succeeded to initialize. (For example the function reports success if the executable is found but the new process fails to initialize because of missing DLLs.)

native raExecuteCmd(cmd[], buf[]="", buflen=0, timeout=-1);

Executes a command with the command line interpreter specified in the COMSPEC environment variable (usually CMD.EXE on Windows NT/2000 and COMMAND.COM on Windows 95/98).

Parameters

cmd

[in] String specifying the command line.

buf

[out] An array that will receive the output produced by the command if buflen is not zero.

buflen

[in] The length of the array specified by buf.

timeout

[in] The time in milliseconds the command waits for command termination before getting output. If -1, it waits for infinity.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

Remarks

This function uses the command line interpreter's output redirection option to store output in a temporary file, so no output redirection should be used in the command line.

If bufLen is set to 0, the function will not wait for the termination of the command.

native raReboot(type=REBOOT_NORMAL);

Initiates reboot.

Parameters

type

[in] One of the following constants defining the type of the reboot process:

REBOOT_NORMAL: normal reboot process (normal termination of running processes, services, etc.)

REBOOT_EMERGENCY: the fastest possible reboot, should be used only in emergency

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

Remarks

After initiating the reboot process, the script should terminate as soon as possible to allow RemotelyAnywhere to shut down correctly.

native raEnumServices();

Takes a snapshot of the currently active services and drivers that can be enumerated.

Remarks

You can call raGetServiceNum, ra.GetService, raGetNextService to retrieve information on the processes.

You must call raEnumServicesClose to release resources allocated by the snapshot.

native raGetServiceNum();

Gets the number of services and drivers in the last snapshot taken with raEnumServices.

native raGetService(*id*, *name*[], *displayName*[], *binary*[], *&type*, *&status*, *&startup*);

native raGetNextService(*name*[], *displayName*[], *binary*[], *&type*, *&status*, *&startup*);

Retrieves information on the specified service or driver in the last snapshot taken with raEnumServices.

Parameters

id

[in] The index of the service or driver to retrieve information on.

name

[out] Array that is being filled with the string representing the short name of the service/driver.

displayName

[out] Array that is being filled with the string representing the long name of the service/driver.

binary

[out] Array that is being filled with the string representing the executable belonging to the service/driver.

type

[out] Variable that receives the type of the service/driver. It can be one of the following constants:

SERVICE_KERNEL_DRIVER: Indicating a device driver

SERVICE_FILE_SYSTEM_DRIVER: Indicating a file system driver

SERVICE_WIN32_OWN_PROCESS: Indicating a service application that runs in its own process

SERVICE_WIN32_SHARE_PROCESS: Indicating a service application that shares a process with other services

SERVICE_INTERACTIVE_PROCESS: Indicating a service application that can interact with the desktop status

[out] Variable that receives the status of the service/driver. It can be one of the following

constants:

- SERVICE_STOPPED: The service is not running
- SERVICE_START_PENDING: The service is starting
- SERVICE_STOP_PENDING: The service is stopping
- SERVICE_RUNNING: The service is running
- SERVICE_CONTINUE_PENDING: The service continue is pending
- SERVICE_PAUSE_PENDING: The service pause is pending
- SERVICE_PAUSED: The service is paused

startup

[out] Variable that receives a value indicating when to start the service/driver. It can be

one of the following values:

- SERVICE_BOOT_START: Started by the system loader
- SERVICE_SYSTEM_START: Started by the IoInitSystem function
- SERVICE_AUTO_START: Started by the service control manager during system startup
- SERVICE_DEMAND_START: Started by the service control manager
- SERVICE_DISABLED: The service/driver cannot be started

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raEnumServicesClose();

Frees resources allocated by raEnumServices.

native raStartService(name[]);

Starts a service/driver.

Parameters

name

[in] String specifying the name of the service to be started.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raStopService(name[]);

Stops a service/driver.

Parameters

name

[in] String specifying the name of the service to be started.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raPauseService(name[]);

Pauses a service/driver.

Parameters

name

[in] String specifying the name of the service to be started.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raContinueService(name[]);

Continues a paused service/driver.

Parameters

name

[in] String specifying the name of the service to be started.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raGetTime();

Retrieves system time.

Return value

Returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

native raGetPerformance(type, time, &value, ...);

Retrieves data from the performance counters managed by RemotelyAnywhere.

Parameters

type

[in] Constant value indicating the type of performance data to retrieve. It can be one of the following:

PERF_DRIVE: Amount of free space on a storage device

PERF_NET_IN: Incoming traffic on a network adapter

PERF_NET_OUT: Outgoing traffic on a network adapter

PERF_CPU_NTH: The appropriate CPU's usage

PERF_CPU: Overall CPU usage

PERF_MEMORY: Memory usage

PERF_PHYSICAL_MEMORY: Physical memory usage

PERF_PAGEFILE: Pagefile usage

PERF_REGISTRY_QUOTA: Registry quota

time

[in] The time when the performance data to be retrieved is registered

time

[out] Variable that will receive the performance data (an integer in the range [0..100]

indicating the usage in percents of the specified resource)

...

[in] Optional variable needed by some type of the performance counters:

PERF_DRIVE: a character indicating which disk drive's performance data are to be retrieved

PERF_NETIN, PERF_NETOUT, PERF_CPU_NTH: An integer value indicating which resource's performance data are to be retrieved

Return value

If the function succeeds, the return value is the period of the performance counter from which the data are retrieved in seconds.

If the function fails, the return value is zero.

native raSleep(time);

Suspends the execution of the script for the specified amount of time.

Parameters

time

[in] The number of milliseconds defining the duration of the suspension.

native raLog(msg[]);

Writes an entry to the RemotelyAnywhere log.

Parameters

msg

[in] The string to be written to the log.

native raSendMail(to[], subj[], msg[], from[]="");

Sends an e-mail message.

Parameters

to

[in] String specifying the addressee of the message.

subj

[in] String specifying the text in the subject of the message.

msg

[in] String specifying the text in message body.

from

[in] String specifying the sender of the message. If an empty string is passed, a default value is used.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

Remarks

The function uses the SMTP server configured in RemotelyAnywhere Configuration/

Miscellaneous.

The function will block until the message is sent or an error occurs.

native raMessage(to[], msg[]);

Sends an administrative message to a user.

Parameters

to

[in] String specifying the user to send the message to.

msg

[in] String specifying the message text.

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.

native raRegGetValue(base, key[], value[], type, ...);

Retrieves a value from the system registry.

Parameters

base

[in] A constant specifying which registry tree to use. It can be one of the following:

HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

key

[in] String specifying the registry key to get the data from.

value

[in] String specifying the name of the value to get.

type

[in] A constant specifying the type of the data. It can be one of the following values:

REG_SZ: String

REG_EXPAND_SZ: String with environment variable references

REG_BINARY: Free form binary (~byte array)

REG_DWORD: 32-bit number

...

Parameters that depend on type:

REG_SZ

REG_EXPAND_SZ

Param5: [out] An array that receives the string

Param6: [in] The length of the array

REG_BINARY

Param5: [out] An array that receives the data

Param6: [in] The maximum number of bytes to receive

REG_DWORD

Param5: [out] A variable that receives the value

Return value

If the function succeeds, the return value is the number of bytes retrieved.

If the function fails, the return value is zero.

```
native raRegSetValue(base, key[], value[], type, ...);
```

Sets a value in the system registry.

Parameters

base

[in] A constant specifying which registry tree to use. It can be one of the following:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_PERFORMANCE_DATA
- HKEY_DYN_DATA

key

[in] String specifying the registry key to write the data to.

value

[in] String specifying the name of the value to write.

type

[in] A constant specifying the type of the data. It can be one of the following values:

- REG_SZ: String
- REG_EXPAND_SZ: String with environment variable references
- REG_BINARY: Free form binary (-byte array)
- REG_DWORD: 32-bit number

...

Parameters that depend on type:

REG_SZ

REG_EXPAND_SZ

Param5: [in] The string to write

REG_BINARY

Param5: [out] An array containing the data to write Param6: [in] The number of bytes to write

REG_DWORD

Param5: [out] The number to write

Return value

If the function succeeds, the return value is 1.

If the function fails, the return value is zero.