SUNWIRE

Solutions that **shine**

SolSwitch Custom Scripting

850 Barrydowne Road, Sudbury, ON P3A 3T7

1-833-727-6777 www.sunwire.ca

Contents

Overview

SolSwitch supports the integration of custom scripts, allowing a low-code implementation of custom functionality that goes beyond the default system features.

Phone calls can be routed to custom scripts in the same way as other system features, and SMS can be configured to run custom scripts upon receiving inbound messages.

Below is a general description of the way that scripts should be written and details on the variables that are predefined by the system based on where the request came from. This document also outlines the operations that are available to be run and their requirements.

Finally, the end of this document will include multiple example scripts for reference.

Script Formatting

The script processing uses line breaks to separate commands, meaning that there should be one command per line and no end of line symbol is required (ie, no semi-colons or other line endings).

Empty lines are ignored, as are any lines starting with **#** or **//**

Variables

Data can be stored in variables, taking the format **${VarName}**, and are set using a single equal sign. Variable names must consist only of letters and numbers. No quotes or concatenation are required, the variable will be set to all text on the line, including the value of other variables. Ex:

```
${a} = Example
${b} = This is an ${a}
```

In this case, the value of ${b} will be set to: *This is an Example*

Data can also be stored in array variables – a single variable which stores multiple values. This can be done by adding square brackets [ ] to the end of a variable, with a key name for the value. Ex:

```
${Var}[Key] = Value
```

For many purposes this functions similar to normal variables, however it is required for some operations such as the CallAPI() function (see below).

Global Variables

${VAR} – This variable will always be set and will contain a text list of all currently set variables. This can be used to log the values of all variables set at any point during a script for troubleshooting or reference purposes. See the Log() function below.

Call Variables

When a call is routed to a custom script from somewhere in the system, the following variables will be pre-defined.

${SrcName} – The name of the caller.

${SrcNum} – The number of the caller. For local calls this will be an extension number, for external calls it will be the DID of the caller.

${DialedNum} – The number that was originally dialed by the caller, either an extension or DID.

${Channel} – The active SIP channel of the call. Can be used for completing various call commands using SolSwitch APIs, see the CallAPI() function below.

SMS Variables

When a script is run by an SMS message, the following variables will be pre-defined. Note: Currently scripts will only be executed when receiving inbound messages, not when sending outbound messages.

${From} – The DID that sent the message, generally an 11 digit number, ex: 17055551234

${To} – The DID on the SolSwitch that received the message, ex: 17055556789

${Body} – The message body of the SMS. Messages are received in single segments, up to a maximum of 160 characters.

Functions

To complete actions within a script, you can make use of functions, which are pre-defined operations that take in data and return values based on the result of the action.

Function calls take the general format of: **FunctionName(Params)**. They can be run on a line on their own, or with a variable to capture the result of the function. Ex:

```
Function(Example 1)

${result} = Function(Example 2)
```

Some functions expect multiple parameters to be passed in the brackets. In these cases, multiple parameters are separated with commas. When passing parameters no quotes are required, however if one of your parameters contains a comma you can use double quotes around the parameter to indicate that it is one parameter. Ex:

```
Function(Parameter 1, "Parameter 2, with a comma")
```

Return Values

When a function returns, it will return the results as an array of values which will be stored in the variable if one is provided.

In the function descriptions below, the array return values will be shown in the format: **[KeyA => ValueA, KeyB => ValueB]**

If the result of a function were stored in the variable ${res} and returned the values above, those values would be accessed as **${res}[KeyA]**. Ex:

```
${res} = Function()
Log(Result: ${res})
Log(Result Key A: ${res}[KeyA])
```

All function calls will return the **Status** key, which will contain **OK** on a success or **FAIL** if an error was encountered. If the status is FAIL, the return will also contain the **Reason** key which will contain details about the error.

For any functions which return values beyond these that will be specified in their individual descriptions below.

General Functions

CallAPI(APIPath)

This function allows the execution of any SolSwitch API. The API must allow access from IP 127.0.0.1 or use a key (in which case the key must be provided).

The parameter for the function must be the path name of the API to call, excluding the solswitch/api/ portion as described by the API documentation. Ex, the URL for the New Call API is as follows: **solswitch/api/newcall/** so a function call to this API would be:

```
CallAPI(newcall)
```

When calling an API, the array variable ${DATA} will automatically be used to populate the attributes for the API request. That means that the attributes required for the request must be added to the ${DATA} array before completing the API call. The only exception to this is that the **Customer** attribute does not need to be provided – it will automatically be set to the customer details of the script.

```
#Place a new call from ext 100 to 705-521-6777

${DATA}[Extension] = 100

${DATA}[Destination] = 17055216777

${DATA}[Key] = HZDLOAXSHTFWIPNQeJEWFJNZKAXQHYKI


CallAPI(newcall)
```

The return values of an API call will be determined by the specific API being called. See the SolSwitch API documentation for additional details on the requirements and return values of each API.

Log(Text)

The log function can be used to create a new log line containing the text passed to the function. Logs can be viewed in the portal at the bottom of the script edit page and are generally used for troubleshooting issues or verifying that the script is operating as expected.

The provided parameter can be text, variables or a combination of both.

```
Log(Logging an example)

Log(${VAR})

Log(Received a new text from ${From} to ${To})
```

Unset(${Var})

The unset function is used to delete a variable or array key/value from the script. Providing the name of an array without a specified key will unset the entire array.

```
Unset(${Var})

Unset(${Var}[Key])
```

Wait(x)

The wait function tells the script to pause for *x* seconds before continuing to the next line of the script. The x value can be an integer or whole number but must be greater than 0.

```
Wait(5)

Wait(1.5)
```

Call Functions

Hangup()

End an ongoing call. Can only be run on a call that has been transferred to the script.

```
Hangup()
```

Playback(Recording)

Play a system recording to an ongoing call. Can only be run on a call that has been transferred to the script. The value of 'Recording' passed to the function should be the name of the recording in *Phone System > System Recordings*.

```
Playback(Generic Welcome)
```

SMS Functions

SendSMS(From, To, Body)

This function will generate a new outbound SMS with the provided details. The From value must be a DID that is configured on the SolSwitch for SMS, and the SMS must allow sending from the IP 127.0.0.1.

**Standard messaging rates apply.*

```
SendSMS(17055216777, 17055551234, Example outbound SMS)
```

Example Scripts

Example 1 – Notify agents of a new message via SMS

Scenario: When a new SMS comes in, send an automated reply to the customer telling them that their message was received, then send messages to a list of agents to notify them that a message is pending.

```
Log("Received new SMS from: ${From}, to: ${To}, body: ${Body}")

#Notify the customer that their message has been received
#From and To are reversed since we want to send to the original from number
SendSMS(${To}, ${From}, "Thank you, your message has been received and an
agent will contact you shortly.")

#Notify agents of the new message
${Msg} = New message from ${From}: ${Body}
SendSMS(${To}, 17055551234, ${Msg})
SendSMS(${To}, 17055555678, ${Msg})
```

Example 2 – Request SMS from IVR Options

Scenario: An IVR is set up with the option 'Press x to request a follow up by text message.'. Pressing that option routes the call to a script to notify agent(s) of the request and tell the customer that their request has been received.

```
#Notify agent
SendSMS(17055216777, 17055551234, Customer ${SrcName} requested SMS
response to ${SrcNum}.)

#Play success recording to user
Playback(SMS Followup Message)

#Wait for recording to complete then end the call if the caller hasn't
Wait(10)
Hangup()
```

Example 3 – Generate new call and transfer caller

Scenario: If a call reaches the overflow voicemail of a queue, queue manager would like to receive a call to alert them. Queue overflow is set to route to a script to call the queue manager while the caller continues to the voicemailbox.

```
#Send the customer's channel to the overflow voicemail
${DATA}[Channel] = ${Channel}
${DATA}[Destination] = 9100
${DATA}[Key] = FzeKKeEKSIUXa7ZRMx3FciZwXG2FW8jd
CallAPI(transfercall)

#Generate call to connect queue manager to alert message
Unset(${DATA}[Channel])
${DATA}[Destination] = 1234
${DATA}[Extension] = 100
CallAPI(newcall)
```