



# ***WPS Communicate*** ***user guide and reference***

Version: 4.1.1

Copyright © 2002-2019 World Programming Limited

[www.worldprogramming.com](http://www.worldprogramming.com)

# Contents

<b>Overview.....</b>	<b>3</b>
<b>Setup and configuration.....</b>	<b>4</b>
<b>Client User Guide.....</b>	<b>6</b>
Key concepts.....	6
Using WPS Communicate.....	6
A simple program.....	6
Using PROC DOWNLOAD.....	8
Using PROC UPLOAD.....	10
Running sub-programs in parallel.....	11
Transferring Macro Variables.....	13
Reference.....	14
How to read EBNF.....	14
Global statements.....	16
Macro processor statements.....	24
WPS Communicate procedures.....	25
System options.....	31
<b>System Administration Guide.....</b>	<b>34</b>
Authentication on z/OS using Telnet.....	34
Sample Telnet authentication script.....	39
SSH (Secure Shell) from a Windows client.....	40
Password authentication (using PuTTY) and WPS sign-on.....	42
Public key authentication.....	47
SSH (Secure Shell) from a UNIX client.....	61
Password authentication and WPS sign-on.....	61
Public key authentication.....	62
Kerberos single sign-on.....	70
<b>Environment variables.....</b>	<b>72</b>
<b>Legal Notices.....</b>	<b>73</b>

# Overview

**WPS Communicate** allows different, selected parts of SAS program logic to be executed on different hosts, and enables the transfer of data and results between them. It can be convenient to think of **WPS Communicate** as enabling programs to move closer to the data on which they operate, unlike the more conventional pattern where data is transferred to the program.

In a normal WPS session, all processing is performed on a single local host - by default, synchronously. It can be advantageous, however, to run different parts of a program on different hosts. For example, you might have a reporting application that needs to extract and summarise some data before generating and distributing reports. It might make sense to execute the extraction and summarisation parts of the program on the host that stores the data, transferring the summarised data back to the local platform for generation and distribution of the reports.

Along with the program code, datasets required for remote processing can be uploaded and downloaded from within a program that is running locally. This makes it possible to perform intensive database work on a central server, before returning to the local platform for the processing of the results.

From version 3.2 onwards, **WPS Communicate** has the capability to operate asynchronously, whereby the initial processing on multiple remote hosts happens in parallel, enabling significant performance improvements for a large class of workloads.

---

**Note:**

Because it is necessary to change the program code to identify the parts that should be run remotely, **WPS Communicate** requires users to be familiar with the language of SAS.

---

This manual is divided into the following parts:

- *Setup and configuration* [↗](#) (page 4).
- The *Client User Guide* [↗](#) (page 6), which is aimed at day-to-day users of **WPS Communicate**.
- The *System Administration Guide* [↗](#) (page 34), which is aimed at the IT people who install and administer the system.

# Setup and configuration

There are 3 items required for a **WPS Communicate** client/server setup: a means of authentication, the WPS software itself, and sufficient WPS license keys to cover the setup.

## Authentication

**WPS Communicate** requires the use of an SSH or Telnet connection between the server and client machines. The available authentication methods, including public key authentication (both with and without the use of a keychain agent) and any additional software required, are described within the *System Administration Guide* [\[link\]](#) (page 34).

The recommended protocol depends on the type of host to which a connection is being made:

- Telnet (refer to *Authentication on z/OS using Telnet* [\[link\]](#) (page 34))

---

**Note:**

For a z/OS host, Telnet (or TN3270) is the most widely-used connection mechanism and is the recommended method for connecting to z/OS hosts using **WPS Communicate**. It is the only supported mechanism that allows direct access to TSO (Time Sharing Option). It is possible to connect to z/OS via SSH (Secure Shell), but this connects you with USS (UNIX System Services) rather than TSO, and this is not always configured on z/OS systems.

---

- SSH (Secure Shell). This is not supplied with the WPS software. However:
  1. For a UNIX (Linux, Solaris or AIX) server machine, the built-in SSH daemon can be used.
  2. For a Windows server machine, the third party SSH facility is available separately from Bitvise (refer to the *System Administration Guide* [\[link\]](#) (page 34) for details).

---

**Note:**

Bitvise SSH is the only SSH facility officially supported by World Programming for a Windows server machine.

---

## WPS software

WPS is supplied as a single installation file that contains all the WPS features required for use with **WPS Communicate**, including **WPS Workbench**, **Java Runtime Environment**, and the licensable **WPS Server** component.

---

**Note:**

You will need WPS installation files that are suitable for the operating systems on both server and client machines. For example, if you have a Linux server machine and Windows client machines, you will need the Linux installation file for the server, and the appropriate Windows installation files (32-bit or 64-bit) for the clients. Refer to the relevant platform installation guide for full details of the WPS installation process.

---

## WPS licence key(s)

In order for a WPS installation to be able to execute a program written in the language of SAS, the **WPS Server** component needs to be activated by the application of a WPS license key (supplied separately to the WPS software).

### Note:

A fully licensed installation of WPS is required on each client and on each host server.

## Installation summary

A brief overview of the sequence of steps required to install and set up a **WPS Communicate** client/server solution is given below.

### Important:

The person who installs WPS and applies the license keys must have operating system administrator privileges on those machines.

1. Outside WPS, set up and test the SSH or Telnet connection between the server and client machines in accordance with the platforms and authentication methods that are active at your site (refer to the *System Administration Guide* [↗](#) (page 34) for details).
2. Install WPS fully on the server machine or z/OS mainframe.

### Note:

Ensure that you have your licence key for the server or z/OS installation. When you launch WPS on a server, you will automatically be prompted to apply the licence. For a z/OS mainframe, you will not be prompted and so you will need to follow the instructions about applying a license key that can be found in the separate document "WPS Installation and User Guide for z/OS".

3. If you have not already done so, install WPS on the individual client machine(s).

### Note:

Ensure that you have your workstation licence key, so that it can be applied to each client in turn. When you launch WPS on the client(s), you will automatically be prompted to apply the licence.

## Environment variables

The configuration of environment variables is not necessary prior to your initial **WPS Communicate** session, but you may want to ask your system administrator to adjust them for subsequent sessions. Refer to *Environment variables* [↗](#) (page 72).

# Client User Guide

## Key concepts

To transfer the flow of control from a program running locally to a remote server, **WPS Communicate** introduced the following two pairs of statements: `SIGNON ... SIGNOFF` and `RSUBMIT ... ENDRSUBMIT`.

`SIGNON` and `SIGNOFF` must enclose the `RSUBMIT` and `ENDRSUBMIT` statements:

```
SIGNON ...;  
...  
RSUBMIT;  
...  
ENDRSUBMIT;  
...  
SIGNOFF;
```

The `SIGNON ...` statement is responsible for initiating and authenticating your session with the server.

Between the `RSUBMIT` and `ENDRSUBMIT` statements, you insert the program code that you intend to run on the remote machine.

Once the remote code has been executed, the `SIGNOFF` statement closes the connection and releases its resources.

## Using WPS Communicate

### A simple program

This section contains an example of a very simple **WPS Communicate** program being executed remotely on a Linux platform.

Before you start, ensure that **WPS Workbench** is installed and running, and that you have successfully signed on to the remote server using an external SSH client.

This program uses simple password sign-on to connect to the remote server. It is not the most secure method, because it stores a user ID and password in plain text in the source code, and the use of Public key - that is to say, password-less - authentication is advised for long-term use (refer to the *System Administration Guide* [↗](#) (page 34)). However, password sign-on does serve to verify that **WPS Communicate** is working in its most basic configuration.

1. Create a new program in **WPS Workbench** as follows:

```
SIGNON <servername> SSH
username="<username>"
password="<password>"
LAUNCHCMD="<path to WPS executable>";
RSUBMIT;
%PUT &SYSHOSTNAME;
%PUT 'Success with simple password sign-on';
ENDRSUBMIT;
SIGNOFF;
```

Be sure to substitute <servername> with the name of your remote server, and, similarly, replace <username> and <password> with your actual user ID and password on the remote machine.

#### Note:

The LAUNCHCMD option needs to point to the location of the WPS executable file on the remote server, for example /home/installs/wps32/bin/wps -dmr.

2. Run the program and examine the output log:

```
367      ODS _ALL_ CLOSE;
368      FILENAME WPSWBHTM TEMP;
369      ODS HTML(ID=WBHTML) BODY=WPSWBHTM GPATH="C:\Users\techwriter\AppData
\Local\Temp\WPS Temporary
369      ! Data\_TD5876";
NOTE: Writing HTML(WBHTML) Body file WPSWBHTM
370      SIGNON DOCSERVER SSH
371      username="XXXX"
372      password=XXXXXXXXXXXX
373      LAUNCHCMD=<path to WPS executable>;
NOTE: Remote SSH signon to DOCSERVER starting
NOTE: Establishing tunnelled connection to DOCSERVER:55765
NOTE: (c) Copyright World Programming Limited 2002-2015. All rights reserved.
NOTE: World Programming System 3.02 (03.02.00.00.011866)
      Licensed to World Programming Company Ltd
NOTE: This session is executing on the Linux platform and is running in 64-bit
mode

NOTE: Remote signon to DOCSERVER complete
374      RSUBMIT;
NOTE: Remote submit to DOCSERVER starting
1      %PUT &SYSHOSTNAME;
      harmony.teamwpc.local
2      %PUT 'Success with simple password sign-on';
      'Success with simple password sign-on'
NOTE: Remote submit to DOCSERVER complete
375      SIGNOFF;
NOTE: Remote signoff from DOCSERVER starting

NOTE: Submitted statements took :
      real time : 0.130
```

```
cpu time : 0.028
NOTE: Remote signoff from DOCSERVER complete
376      quit; run;
377      ODS _ALL_ CLOSE;
```

- The `SSH`, `username`, `password` and `LAUNCHCMD` options of the `SIGNON` statement provide all the information necessary to log onto the remote host via SSH and initiate the remote WPS server.
- Between the `RSUBMIT` and the `ENDRSUBMIT` statements, the following two lines of code are executed on the remote machine:

```
%PUT &SYSHOSTNAME;
%PUT 'Success with simple password sign-on';
```

### Note:

The `%PUT` statements write to the local log file, but the `&SYSHOSTNAME` macro variable is resolved on the remote machine (to the `DOCSERVER` in this case).

- Finally, the `SIGNOFF` statement deletes our connection.

## Using PROC DOWNLOAD

`PROC DOWNLOAD` is intended to transfer libraries, datasets or files from a remote host, using the options `INLIB`, `DATA` and `INFILE` respectively, to the local platform, using the options `OUTLIB`, `OUT` and `OUTFILE`. `PROC DOWNLOAD` must be placed inside an `RSUBMIT ... ENDRSUBMIT` block of code.

The following are some syntax examples:

```
/* transfer a library */
PROC DOWNLOAD INLIB=remotelib OUTLIB=locallib; RUN;
```

```
/* transfer a single dataset */
PROC DOWNLOAD DATA=remotelib.dataset OUT=locallib.dataset; RUN;
```

```
/*transfer a file */
PROC DOWNLOAD INFILE="remote_host_file_path" OUTFILE="local_platform_file_path";
RUN;
```

### Note:

For the full syntax for this procedure, please refer to the **Reference** section.

## Example of the use of PROC DOWNLOAD

This section contains an example using `PROC DOWNLOAD`.

1. Create a new program in **WPS Workbench** by copying and pasting the following code:

```
/******
Sign on to the remote host
```



```

*****/
signon <servername> ssh
user='<username>'
password='<password>'
launchcmd='<path to WPS executable>';

/*****
Create and populate a small dataset on the remote host
*****/
rsubmit;
data communicatedemo;
input movie $ 1-46 year $ 48-51;
cards;
Avatar                                2009
Titanic                              1997
The Avengers                          2012
Harry Potter and the Deathly Hallows - part 2  2011
Frozen                                2013
;
run;

/*****
Download the generated dataset to the local platform
*****/
proc download
data=WORK.communicatedemo
out=WORK.communicatedemo;
run;

/*****
Complete the remote program execution and close the connection
*****/
endrsubmit;
signoff;

```

Be sure to substitute `<servername>` with the name of your remote server, replace `<username>` and `<password>` as appropriate, and configure `launchcmd` to point to the location of the WPS executable file on the remote server, for example `/home/installs/wps-3.2/bin/wps -dmr`. This program is intended to create a dataset on the remote host, listing 5 of the top-grossing films of all time, and download it to your local platform.

2. Run the program and examine the output log - a message beneath the `PROC DOWNLOAD` invocation explains the data transfer:

```

NOTE: Dataset download in progress from WORK.communicatedemo to
      WORK.communicatedemo
NOTE: 333 bytes were transferred to dataset WORK.communicatedemo at 333000 bytes/
sec
NOTE: Dataset "WORK.communicatedemo" has 5 observation(s) and 2 variable(s)

```

The program runs, creating the dataset on the remote machine and downloading it to the local platform. You can examine the `communicatedemo` dataset in your local `WORK` library by selecting **Local Server > Libraries > Work** in the **WPS Server Explorer** tab.

## Using PROC UPLOAD

PROC UPLOAD is intended to transfer libraries, datasets or files from the local platform, using the options INLIB, DATA and INFILE respectively, to a remote host, using the options OUTLIB, OUT and OUTFILE. PROC UPLOAD must be placed inside an RSUBMIT ... ENDRSUBMIT block of code.

The following are some syntax examples:

```
/* transfer a library */
PROC UPLOAD INLIB=locallib OUTLIB=remotelib; RUN;
```

```
/* transfer a single dataset */
PROC UPLOAD DATA=locallib.dataset OUT=remotelib.dataset; RUN;
```

```
/*transfer a file */
PROC UPLOAD INFILE="local_host_file_path" OUTFILE="remote_host_file_path"; RUN;
```

---

**Note:**

For the full syntax for this procedure, please refer to the **Reference** section.

---

## Example of the use of PROC UPLOAD

This section contains an example using PROC UPLOAD, wherein the dataset that was downloaded via the PROC DOWNLOAD example is uploaded and saved to a second remote host.

1. Copy and paste the following code:

```
/******
Sign on to the remote host to upload the dataset
******/
signon <server2name> ssh
user='<username>'
password='<password>'
launchcmd='<path to WPS executable>';

/******
Create a library for the dataset on the host
******/
rsubmit;
libname rlib "/home/<username>/datasets";

/******
Upload the dataset to the host and output its contents to the library
******/
proc upload
data=WORK.communicatedemo
out=rlib.communicatedemo;
run;

/******
Complete the remote program execution
******/
```

```
endrsubmit;

/*****
Sign off the remote host and close the connection
*****/
signoff;
```

Be sure to substitute `<server2name>` with the name of your second remote server, replace `<username>` and `<password>` as appropriate, and configure `launchcmd` to point to the location of the WPS executable file on the remote server, for example `/home/installs/wps-3.2/bin/wps-dmr`.

### Important:

Once the `signoff` statement is encountered, all the datasets in the `WORK` location on the remote host will be removed because `WORK` is a temporary location. If you do not wish this to happen, you should output the dataset to a permanent location, by, for example, using a library such as `rlib` in the above code.

2. Run the program and examine the output log - a message beneath the `PROC UPLOAD` invocation explains the data transfer:

```
NOTE: Dataset upload in progress from WORK.communicatedemo to
      WORK.communicatedemo
NOTE: 333 bytes were transferred to dataset WORK.communicatedemo at 333000 bytes/
      sec
NOTE: Dataset "WORK.communicatedemo" has 5 observation(s) and 2 variable(s)
```

The program runs and uploads the dataset.

### Important:

You can only examine the dataset if you saved it to a permanent location (that is to say, not `WORK`), by, for example, running **WPS Workbench** on the remote server and executing a library opening statement such as `LIBNAME rlib "/home/<username>/datasets"`. It will then appear in the **WPS Server Explorer** tab under **Libraries**.

## Running sub-programs in parallel

### Asynchronous WPS Communicate

**WPS Communicate** can run sub-programs asynchronously - that is to say, one sub-program does not have to wait for the completion of another before it is executed. Depending on the relative workloads of the separate sub-programs, this can lead to significant performance improvements - waiting is reduced from the sum of the durations of the sub-programs to the length of the longest sub-program.

Whether a remote sub-program is run synchronously or asynchronously is controlled by the `WAIT` option of its invoking `RSUBMIT` statement, and a corresponding `WAITFOR` statement which is used to make WPS wait for one or more remote sub-programs to complete.

In the following code fragment, sub-programs are run in parallel on *host1* and *host2*. When they have finished, processing continues locally. Such processing might, for example, perform a merge of the results of the two earlier sub-programs.

```
%let remote-id1 = host1
%let remote-id2 = host2
/*****
Sign on to the remote machines
*****/
signon <remote-id1> ssh
user='<user1>'
password='<password1>'
launchcmd='<wps-install-location>/bin/wps -dmr';

signon <remote-id2> ssh
user='<user2>'
password='<password2>'
launchcmd='<wps-install-location>/bin/wps -dmr';

/*****
Execute sub-programs
*****/
rsubmit <remote-id1> wait=no;
/****
Run code on <remote-id1>
****/
endrsubmit;

rsubmit <remote-id2> wait=no;
/****
Run code on <remote-id2>
****/
endrsubmit;

/*****
Wait for all remote processing to complete
*****/
WAITFOR _ALL_ <remote-id1> <remote-id2>;

/*****
Release connections to remote machines
*****/
signoff <remote-id1>;
signoff <remote-id2>;

/*****
Perform final local processing
*****/
data _null_;
/****
Local data step processing
****/
run;
```

The WAIT=NO option of both RSUBMIT statements informs WPS that the sub-programs should be executed asynchronously.

**Note:**

The `WAITFOR _ALL_` statement causes the main program to suspend execution until processing is complete on **all** of the server `remote-ids`, or until the `TIMEOUT` interval, if specified, has expired. If you use `WAITFOR _ANY_ <remote-id1> <remote-id2>`, or simply `WAITFOR <remote-id1> <remote-id2>`, instead of `WAITFOR _ALL_`, then the main program will suspend execution until processing is complete on **any** of the server `remote-ids` (or until the `TIMEOUT` interval, if specified, has expired). The default is `_ANY_` rather than `_ALL_` if no argument is supplied between `WAITFOR` and the server `remote-ids`.

## Running local sub-programs in parallel

**WPS Communicate** offers an additional way to parallelise local processing by enabling you to create multiple connections to your local platform just as if they were connections to remote hosts. Instead of providing full sign-on statements requiring authentication, as described above, it is sufficient to use:

```
signon local1 launchcmd="<local-wps-install-directory>\bin\wps.exe -dmr";  
signon local2 launchcmd="<local-wps-install-directory>\bin\wps.exe -dmr";
```

or simply

```
signon local1 launchcmd="!sascmd -dmr"  
signon local2 launchcmd="!sascmd -dmr"
```

The `local1` and `local2` arguments become aliases for connections to your local platform that can be used in exactly the same way as the host names of remotely connected servers. Although it is dependent upon the resource usage patterns of the various sub-programs, such a technique can lead to improvements in performance even though there is no net increase in available CPU or I/O bandwidth.

## Transferring Macro Variables

Macro variables can be passed between the local platform and a remote host, using the `%SYSLPUT` and `%SYSRPUT` macro processor statements.

The `%SYSLPUT` statement creates a macro variable on a remote host with which you have established a **WPS Communicate** session. The macro call must be placed after the `SIGNON` statement, but before the `RSUBMIT` statement.

The `%SYSRPUT` statement retrieves a macro variable from a remote host to which there is an established **WPS Communicate** session, creating an identical local macro variable. The macro call can only be placed inside an `RSUBMIT ... ENDRSUBMIT` block of code because it is being executed on the remote host and returns variables back to the local platform.

The following are some examples of code snippets using these calls:

```
signon <servername> ssh  
user='<username>'  
password='<password>'  
launchcmd='<path to WPS executable>';
```

```
/******  
Send over a macro definition from the local to the remote platform  
******/  
%SYSLPUT LOCALOS=&SYSSCPL.;  
  
/******  
SUBMIT the following code to UNIX, between the RSUBMIT/ENDRSUBMIT block  
******/  
rsubmit;  
  
%put "EXECUTING ON REMOTE OS &SYSSCPL. FROM LOCAL OS &LOCALOS.";  
%put &SYSHOSTNAME;  
  
%SYSRPUT REMOTEOS=&SYSSCPL.;  
  
endrsubmit;  
  
%put "EXECUTING ON LOCAL OS &SYSSCPL. FROM REMOTE OS &REMOTEOS.";  
  
/******  
SIGNOFF the UNIX platform, preventing further RSUBMITs  
******/  
signoff;  
  
%put &SYSHOSTNAME;
```

**Note:**

For the full syntax for these macro processor statements, please refer to the **Reference** section.

## Reference

EBNF (Extended Backus-Naur Form) definitions are notations that help to explain the syntax of programming languages, and they are used in this guide to describe the language syntax.

## How to read EBNF

EBNF is a textual notation that accompanies significant language structures such as procedures, statements and so on.

The description of each language concept commences with its EBNF representation .

### Entering text

Text that should be entered exactly as displayed is shown in a typewriter font :

OUTPUT ;

This example describes a fragment of syntax in which the keyword `OUTPUT` is followed by a semi-colon character: `;`. The EBNF version of this fragment is simply the characters: `OUTPUT;`.

Generally the case of the text is not significant, but in this reference, it is the convention to use upper-case for keywords.

## Placeholder items

Placeholders that should be substituted with relevant, context-dependent text are rendered in lower case, surrounded by a box :

```
OUTPUT data-set-name ;
```

Here, the keyword `OUTPUT` should be entered literally, but *data-set-name* should be replaced by something appropriate to the program – in this case, the name of a dataset to add an observation to.

## Optionality

When items are optional, they appear in square brackets `[ ]` in EBNF :

```
OUTPUT [data-set-name] ;
```

## Repetition

In EBNF, repetition is denoted by curly braces and ellipsis `{ ... }`. An optional separator for multiple instances is shown just before the ellipsis.

```
OUTPUT { data-set-name ... } ;
```

Above, the keyword `OUTPUT` should be entered literally, and it should be followed by one or more repetitions of *data-set-name* - in this case, no separator other than a space has been asked for.

The example below shows the use of a separator.

```
function-name ( { argument, ... } ) ;
```

## Choices

In EBNF, the choice is represented by a logical OR symbol `|` between each option.

```
GETNAMES [ YES | NO ] ;
```

In the above example, the keyword `GETNAMES` should be entered literally, and then either the keyword `YES` or the keyword `NO`.

## Fragments

When the syntax is too complicated to fit in one definition, it might be broken into fragments:

```
PROC PRINT { option ... }
```

## option

[ DATA = data-set-name | LABEL ]

Above, the whole syntax is split into separate EBNF fragments. The first indicates that `PROC PRINT` should be followed by one or more instances of an *option*, each of which must adhere to the syntax given in the second diagram.

## Help with reading EBNF

The table below summarises the EBNF conventions used in this reference:

Convention	Description
[ ]	Optional elements
{ ... }	Repeated group of elements
{ , ... }	Separators that enclose iterations of repeated items are shown just before the ellipsis
	Either/or within a selection group enclosed in [ ]. For example: <ul style="list-style-type: none"><li>• [ A   B ] – choose either A or B</li><li>• [ A   B   ] – choose A or B or nothing (i.e. omit the group altogether)</li></ul>
" "	Comment – for example, a group title
monospaced	Keywords and separators
<i>italics</i>	variables
<b>bold</b>	Elements presented in their own diagrams, like fragments
<u>underlined</u>	Default selection

## Global statements

### ENDRSUBMIT

```
ENDRSUBMIT ;
```

This statement indicates the end of a block of code that began with an `RSUBMIT` statement.



## RSUBMIT

```
RSUBMIT [ remote-id ] [ { option ... } ] ;
```

### option

[ *options A to M* | *options N to Z* ]

### options A to M

```
[ [ CMACVAR | MACVAR ] = 'variable-name' |
  [ CONNECTPERSIST | CPERSIST | PERSIST ] = [ YES | NO ] |

  [ CONNECTREMOTE |
    CREMOTE |
    REMOTE |
    PROCESS
  ] = remote-id |
  [ CONNECTWAIT | CWAIT | WAIT ] = [ YES | NO ] |
  [ CSYSRPUTSYNC | SYSRPUTSYNC ] = [ YES | NO ] |
  [ CSCRIPT | SCRIPT ] = signon-script |
  IDENTITYFILE = identity-file |
  LOG = [ KEEP | PURGE | filename ]
]
```

### options N to Z

```
[ [ NOCSCRIPT | NOSCRIPT ] |
  OUTPUT = [ KEEP | PURGE | filename ] |

  [ PASSWORD |
    PASS |
    PASSWD |
    PW |
    PWD
  ] = string |
  [ SASCMD | LAUNCHCMD ] = 'command-name' |
  SIGNONWAIT = [ YES | NO ] |
  TBUFSIZE = [ bytes | kilobytes K | megabytes M ] |

  [ UID |
    USER |
    USERID |
    USERNAME
  ] = 'string' |
```

```
SSH |  
DEBUG
```

```
]
```

This statement marks the beginning of a block of program code to be submitted to a (usually remote) host for execution.

## CMACVAR, MACVAR

This option specifies a macro variable whose value is bound to the completion status of the current RSUBMIT block.

## CONNECTPERSIST, CPERSIST, PERSIST

This option signifies whether or not an automatic signoff occurs after a SIGNON and RSUBMIT.

## CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option identifies the remote machine to which a connection will be established, either directly or by naming a macro variable that contains the address.

---

### Note:

If the `CONNECTREMOTE` option is used with the name of the remote host specifically provided as a macro variable, then no ampersand should be placed before the macro variable name. The correct syntax is illustrated in the following fragment:

```
...  
%LET HostName = RemoteHost;  
  
options ssh_hostvalidation=none;  
signon connectremote=HostName ssh /* Not &HostName */  
user = <username>  
password = <password>  
launchcmd = '<location-of-wps-executable> -dmr';  
...
```

## CONNECTWAIT, CWAIT, WAIT

This option determines if the RSUBMIT block is to be run in asynchronous or synchronous mode, by setting it to NO or YES respectively.

## CSYSRPUTSYNC, SYSRPUTSYNC

If set to YES, this option forces macro variables to be defined when %SYSRPUT executes.

## CSCSCRIPT, SCRIPT

This option identifies a signon script.

## IDENTITYFILE

This option specifies a file containing authentication information, such as SSH keys.

## NOSCRYPT, NOCRYPT

This option indicates that no script should be used to sign on.

## LOG

This option defines whether the system log should be kept, purged or sent to a specific file.

## OUTPUT

This option defines whether the output of the sub-program should be kept, purged or sent to a specific file.

## PASSWORD, PASS, PASSWD, PW, PWD

This option is used to specify a password for remote authorisation.

## SASCMD, LAUNCHCMD

When present, this option is used to specify the command required to launch WPS on the remote machine.

## SIGNONWAIT

This option stipulates that a `SIGNON` should finish before permitting subsequent processing.

## TBUFSIZE

This option specifies the WPS COMMUNICATE message buffer size.

## UID, USER, USERID, USERNAME

When present, this option specifies the user name.

## SSH

This option specifies that the connection will utilise the encrypted SSH protocol.

## DEBUG

This option specifies that extra debugging messages are written to the system log.

## SIGNOFF

```
SIGNOFF [ [ remote-id ] [ _ALL_ ] [ { option ... } ] ;
```

### option

```
[ [ CMCVAR | MACVAR ] = 'variable-name' |  
[ CONNECTREMOTE |  
  CREMOTE |  
  REMOTE |  
  PROCESS  
] = remote-id |  
[ CSCRIPT | SCRIPT ] = signon-script |  
[ NOCSCRIPT | NOSCRIPT ]  
]
```

This statement closes down a connection with a remote server, following the execution of a remotely executed block of code.

### CMACVAR, MACVAR

This option specifies a macro variable associated with the remote session and whose value is bound to the completion status of the current `SIGNOFF` statement.

### CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option names the remote session from which you wish to sign off.

### CSCRIPT, SCRIPT

This option identifies a script to be executed during signoff.

### NOSCRIPT, NOSCRIPT

This option indicates that no script should be involved in the signoff process.

## SIGNON

```
SIGNON [ remote-id ] [ { option ... } ] ;
```

### option

```
[ options A to O | options P to Z ]
```

options **A** to **O**

```
[ [ CMACVAR | MACVAR ] = 'variable-name' |

    [ CONNECTREMOTE |
      CREMOTE |
      REMOTE |
      PROCESS
    ] = remote-id |
  [ CONNECTWAIT | CWAIT | WAIT ] = [ YES | NO ] |
  [ CSCRIPT | SCRIPT ] = signon-script |
  [ CSYSRPUTSYNC | SYSRPUTSYNC ] = [ YES | NO ] |
  IDENTITYFILE = identity-file |
  LOG = [ KEEP | PURGE | filename ] |
  [ NOCSCRIPT | NOSCRIPT ] |
  OUTPUT = [ KEEP | PURGE | filename ]
]
```

options **P** to **Z**

```
[
    [ PASSWORD |
      PASS |
      PASSWD |
      PW |
      PWD
    ] = string |
  [ SASCMD | LAUNCHCMD ] = 'command-name' |
  SIGNONWAIT = [ YES | NO ] |
  TBUFSIZE = [ bytes | kilobytes K | megabytes M ] |

    [ UID |
      USER |
      USERID |
      USERNAME
    ] = 'string' |
  SSH |
  DEBUG
]
```

This statement and its options provide the information necessary to specify where the remote WPS installation is located, plus credentials to connect and log in to the remote server, prior to invoking a block of remotely executed code.

## CMACVAR, MACVAR

This option specifies a macro variable associated with the remote session and whose value is bound to the completion status of the current `SIGNON` statement.

## CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option names the remote session.

Note that if the `CONNECTREMOTE` option is used with the name of the remote host specifically provided as a macro variable, then (perhaps counterintuitively) no ampersand should be placed before the macro variable name. The correct syntax is illustrated in the following fragment:

```
...
%LET HostName = RemoteHost;

options ssh_hostvalidation=none;
signon connectremote=HostName ssh /* Not &HostName */
user = <username>
password = <password>
launchcmd = '<location-of-wps-executable> -dmr';
...
```

## CONNECTWAIT, CWAIT, WAIT

This option determines if the `RSUBMIT` block is to be run in asynchronous or synchronous mode, by setting it to `NO` or `YES` respectively.

## CSYSRPUTSYNC, SYSRPUTSYNC

If set to `YES`, this option forces macro variables to be defined when `%SYSRPUT` executes.

## CSCSCRIPT, SCRIPT

This option identifies a signon script.

## IDENTITYFILE

This option specifies a file containing authentication information, such as SSH keys.

## NOSCRIPT, NOCSCRIPT

This option indicates that no script should be used to sign on.

## LOG

This option defines whether the system log should be kept, purged or sent to a specific file.

## OUTPUT

This option defines whether the output of the sub-program should be kept, purged or sent to a specific file.

## PASSWORD, PASS, PASSWD, PW, PWD

This option is used to specify a password for remote authorisation.

## SASCMD, LAUNCHCMD

When present, this option is used to specify the command required to launch WPS on the remote machine.

## SIGNONWAIT

This option stipulates that a `SIGNON` should finish before permitting subsequent processing.

## TBUFSIZE

This option specifies the WPS COMMUNICATE message buffer size.

## UID, USER, USERID, USERNAME

When present, this option specifies the user name.

## SSH

This option specifies that the connection will utilise the encrypted SSH protocol.

## DEBUG

This option specifies that extra debugging messages are written to the sytem log.

## WAITFOR

```
WAITFOR [ | _ANY_ | _ALL_ ] { remote-id ... } [ TIMEOUT = seconds ] ;
```

In that the above diagram applies to WPS Communicate only, the `WAITFOR _ALL_` statement suspends execution of the current session until processing is complete for **all** of the server `remote-ids`, or until the `TIMEOUT` interval, if specified, has expired.

If you use `WAITFOR _ANY_`, or simply `WAITFOR`, instead of `WAITFOR _ALL_`, then execution of the session will only be suspended until processing is complete on one of the server `remote-ids` (or until the `TIMEOUT` interval, if specified, has expired).

**Note:**

As implied above, the default is `_ANY_` rather than `_ALL_` if no argument is supplied between `WAITFOR` and the `remote-ids`.

## Macro processor statements

These statements enable you to create and retrieve the value of a macro variable on a remote server.

### %SYSLPUT

```
%SYSLPUT  
[ macro-variable = value |  
  _ALL_ |  
  _AUTOMATIC_ |  
  _GLOBAL_ |  
  _LOCAL_ |  
  _USER_  
][ / { option ... } ] ;
```

#### option

```
[ LIKE = 'pattern' | REMOTE = remote-id ]
```

This statement creates a macro variable on a remote host with which you have established a WPS Communicate session. It should be placed outside of the corresponding `RSUBMIT` block.

### %SYSRPUT

```
%SYSRPUT macro-variable = value ;
```

This statement retrieves a macro variable from a remote host to which there is an established WPS Communicate session, creating an identical local macro variable. It should be placed inside the corresponding `RSUBMIT` block.



# WPS Communicate procedures

These procedures enable you to transfer files, libraries or datasets to and from a remote host.

## DOWNLOAD Procedure

This procedure downloads one or more files, libraries or datasets from a remote host. It can only be invoked from inside an `RSUBMIT` block.

### Supported statements

- `PROC DOWNLOAD` [↗](#) (page 25)
- `EXCLUDE` [↗](#) (page 27)
- `SELECT` [↗](#) (page 27)
- `WHERE` [↗](#) (page 28)

## PROC DOWNLOAD

`PROC DOWNLOAD [ { option ... } ] ;`

### option

```
[ AFTER = numeric |
  BINARY |
  DATA = server-data-set [ ( dataset-options ) ] |
  DATECOPY |
  EXTENDSN = [ NO | YES ] |
  INDEX = [ NO | YES ] |
  INFILE = server-file-reference |
  [ INLIB | IN | INDD ] = server-library-name |
  MEMTYPE =
    [ ALL |
      CATALOG |
      DATA |
      MDDB |
      VIEW
    ] |
  OUT = [ library.dataset | dataset ] [ ( dataset-options ) ] |
  OUTFILE = client-file-reference |
  [ OUTLIB | OUTDD | OUT ] = client-library-name |
  V6TRANSPORT
```

## 1

**AFTER**

Specifies a numeric modification date, ensuring that only datasets or libraries modified after this date are downloaded. This option is invalid for external file downloads.

**BINARY**

Valid only when downloading external files, this option specifies that the transfer should be an exact, binary copy.

**DATA**

Specifies the name of a dataset to be downloaded.

**DATECOPY**

When present, this option indicates that a remote dataset's creation date and time should be retained when it is downloaded. This option is invalid for external file downloads.

**EXTENDSN**

Specifies if short numeric variables should have their lengths extended. This option is invalid for external file downloads and might be considered if transferring datasets from a mainframe to a PC.

**INDEX**

For remote datasets that have indexes, this indicates whether these indexes should be re-established on the local machine after the download. This option is invalid for external files downloads.

**INFILE**

Specifies the name of a remote external file to download. If this option is present, so must the `OUTFILE=` option be.

**INLIB**

Specifies the name of the remote library. This option is invalid for external file downloads.

**OUT**

Specifies the name of the receiving local dataset. This option is invalid for external file downloads.

**OUTFILE**

Specifies the name of local file to receive an external file download. If this option is present, so must the `INFILE` option be.

## OUTLIB

Specifies the name of the local library into which a remote dataset is downloaded. This option is invalid for external file downloads.

## V6TRANSPORT

This is a translation option when exchanging data between two different versions.

## EXCLUDE

```
EXCLUDE { data-set ... } [ / option ] ;
```

### data-set

```
data-set-name [ : ]  
[ ( [ MEMTYPE | MTYPE | M ] = [ DATA | VIEW | CATALOG ] )  
]
```

### option

```
MEMTYPE = [ DATA | VIEW | CATALOG | ALL ]
```

## MEMTYPE

This option specifies the member types to be downloaded - see the syntax diagram above. This option is invalid for external file downloads.

## SELECT

```
SELECT { data-set ... } [ / option ] ;
```

### data-set

```
data-set-name [ : ]  
[ ( [ MEMTYPE | MTYPE | M ] = [ DATA | VIEW | CATALOG ] )  
]
```

### option

```
MEMTYPE = [ DATA | VIEW | CATALOG | ALL ]
```

## MEMTYPE

See statement `EXCLUDE`.

## WHERE

Restricts the observations to be processed.

```
WHERE condition ;
```

## UPLOAD Procedure

This procedure uploads one or more files, libraries or datasets to a remote host. It can only be invoked from inside an `RSUBMIT` block.

### Supported statements

- `PROC UPLOAD` [↗](#) (page 28)
- `EXCLUDE` [↗](#) (page 30)
- `SELECT` [↗](#) (page 30)
- `WHERE` [↗](#) (page 31)

## PROC UPLOAD

```
PROC UPLOAD [ { option ... } ] ;
```

### option

```
[ AFTER = numeric |  
  BINARY |  
  DATA = client-data-set |  
  DATECOPY |  
  EXTENDSN = [ NO | YES ] |  
  INDEX = [ NO | YES ] |  
  INFILE = client-file-reference |  
  [ INLIB | IN | INDD ] = client-library-name |  
  MEMTYPE =  
    [ ALL |  
      CATALOG |  
      DATA |  
      Mddb |
```

```

VIEW
] |
OUT = [ library.dataset | dataset ] ( dataset-options ) ] |
[ OUTLIB | OUTDD | OUT ] = server-library-name |
OUTFILE = server-file-reference |
V6TRANSPORT
]

```

## AFTER

Specifies a numeric modification date, ensuring that only datasets or libraries modified after this date are uploaded. This option is invalid for external file uploads.

## BINARY

Valid only when uploading external files, this option specifies that the transfer should be an exact, binary copy.

## DATA

Specifies the name of a dataset to be uploaded.

## DATECOPY

When present, this option indicates that a local dataset's creation date and time should be retained when it is uploaded. This option is invalid for external file uploads.

## EXTENDSN

Specifies if short numeric variables should have their lengths extended. This option is invalid for external file uploads and might be considered if transferring datasets to a mainframe from a PC.

## INDEX

For local datasets that have indexes, this indicates whether these indexes should be re-established on the remote machine after the upload. This option is invalid for external files uploads.

## INFILE

Specifies the name of a local external file to upload. If this option is present, so must the `OUTFILE=` option be.

## INLIB

Specifies the name of the local library. This option is invalid for external file uploads.

## MEMTYPE

This option specifies the member types to be uploaded - see the syntax diagram above. This option is invalid for external file uploads.

## OUT

Specifies the name of the receiving remote dataset. This option is invalid for external file uploads.

## OUTFILE

Specifies the name of remote file to receive an external file upload. If this option is present, so must the `INFILE` option be.

## OUTLIB

Specifies the name of the remote library into which a local dataset is uploaded. This option is invalid for external file uploads.

## V6TRANSPORT

This is a translation option when exchanging data between two different versions.

## EXCLUDE

```
EXCLUDE { data-set ... } [ / option ] ;
```

### data-set

```
data-set-name [ : ]
[ ( [ MEMTYPE | MTYPE | M ] = [ DATA | VIEW | CATALOG ] )
]
```

### option

```
MEMTYPE = [ DATA | VIEW | CATALOG | ALL ]
```

## SELECT

```
SELECT { data-set ... } [ / option ] ;
```

## data-set

```
data-set-name [ : ]  
[ ( [ MEMTYPE | MTYPE | M ] = [ DATA | VIEW | CATALOG ] )  
]
```

## option

```
MEMTYPE = [ DATA | VIEW | CATALOG | ALL ]
```

## WHERE

Restricts the observations to be processed.

```
WHERE condition ;
```

## System options

### AUTOSIGNON

```
[ AUTOSIGNON | NOAUTOSIGNON ]
```

Valid in:                      `OPTIONS` statement, configuration file and command line

Default:                      NOAUTOSIGNON

### Description

When this system option is active, remote submit will attempt to automatically sign on.

### COMAMID

```
COMAMID = communication-method
```

Valid in:                      `OPTIONS` statement, configuration file and command line

Default:                      TCP

Max length:                  8

## Description

This system option specifies the communication method to use for establishing remote communications.

## CONNECTPERSIST

[ CONNECTPERSIST | NOCONNECTPERSIST ]

Valid in: OPTIONS statement, configuration file and command line

Default: CONNECTPERSIST

## Description

When set, this system option specifies that a remote connection will be persisted after an `RSUBMIT` block. This system option is an alias of `CPERSIST`.

## CONNECTREMOTE

CONNECTREMOTE = server-name

Valid in: OPTIONS statement, configuration file and command line

Default: *blank*

Max length: 1024

## Description

This system option identifies a specific remote server to connect to. It is blank (empty string) by default.

## DMR

[ DMR | NODMR ]

Valid in: Command line only

Default: NODMR

## Description

This system option invokes a WPS COMMUNICATE server session. It is inactive by default and can be effected only via the command line.



## SASCMD

SASCMD = `command`

Valid in:                    `OPTIONS` statement, configuration file and command line  
 Default:                    ""  
 Max length:                32767

### Description

This system option specifies the command to be used by WPS COMMUNICATE to start another local WPS session. It is blank (empty string) by default.

## SASSCRIPT

SASSCRIPT = ( { `location` ... } )

Valid in:                    `OPTIONS` statement, configuration file and command line  
 Default:                    ""  
 Max length:                1024

### Description

This system option specifies the location of the WPS COMMUNICATE signon scripts. It is blank (empty string) by default.

# System Administration Guide

This part of the guide is intended for system administrators who are responsible for the server authentication relating to **WPS Communicate** and **WPS Link**, and for the generation and deployment of any required public and private keys.

**Note:**

**WPS Communicate** and **WPS Link** are separate features that can run independently. That is to say, you do not need one in order to run the other. You would use **WPS Communicate** to run selective code on remote server hosts or a z/OS mainframe, and **WPS Link** to run entire programs, via the Workbench GUI, on remote server hosts only. The two features are described together here as they both require means of remote authentication.

The following is a summary of the authentication methods as they apply to both **WPS Communicate** and **WPS Link**.

Authentication Method	WPS Communicate	WPS Link
Password	Yes	Yes
Public key with passphrase and keychain agent	Yes	Yes
Public key with passphrase and no keychain agent	No	Yes
Kerberos	Yes	Yes
Telnet on z/OS	Yes	No

## Authentication on z/OS using Telnet

**WPS Communicate** can use a Telnet sign-on to a remote z/OS host to launch a WPS server via a supplied `CLIST` called `TSOWPS`. No USS (UNIX System Services) configuration is required, and the USS portion of WPS is unused.

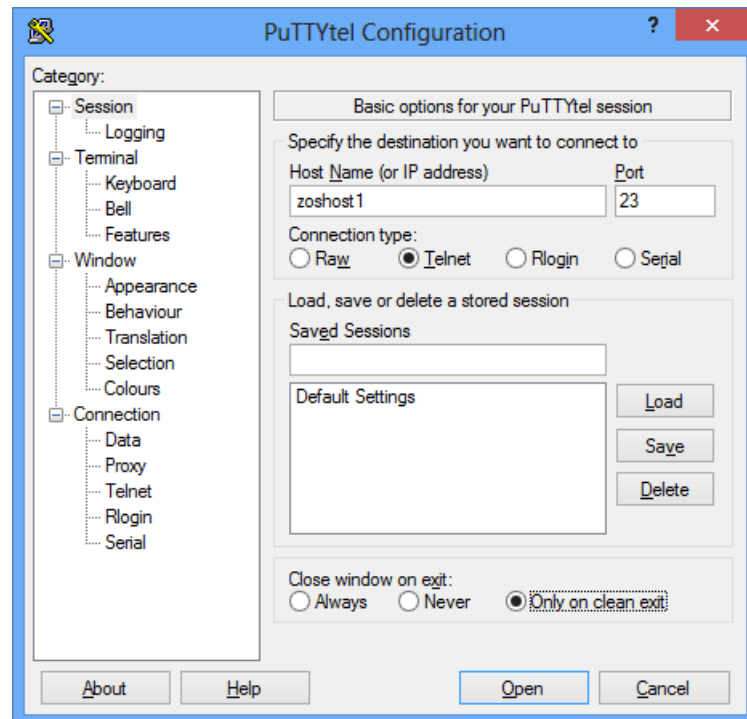
1. First, ensure that the `TSOWPS CLIST` is operational for your installation, by performing a normal TN3270 logon using a 3270 terminal emulator to your z/OS host and invoking the supplied `TSOWPS CLIST`. Typically, you might use ISPF ((Interactive System Productivity Facility) Option 6 and type `TSOWPS` at the prompt.

If you receive a message similar to the following, then you may need to make changes to the `TSOWPS CLIST` or your WPS installation:

```
WPS CANNOT BE INVOKED AS WPSPFX HAS NOT BEEN DEFINED
```

This message informs you that the installation dataset prefix is unresolved and that WPS is unable to locate its various components.

2. Capture the Telnet sign-on prompts by performing a manual sign-on to the z/OS host using a Telnet client such as the PuTTYtel client. Start this client and enter the name of the host in the **Host Name (or IP address)** entry field. Ensure that the **Telnet** radio button is selected:



Subsequent prompts will depend on how your z/OS host has been set up. The purpose of this step is twofold:

- The first is to verify that you can perform a Telnet connection to the host.
- The second is to capture the prompts and any required responses.

The following log captures a typical sequence of prompts when performing a Telnet login to a z/OS host. Our goal is to understand this sequence, and then use it to help write a script that will automate the process.

#### Note:

The user's entered responses have not been echoed to this output for the reason that the exact sequence of inputs and responses is often installation-dependent. It follows that the hand-coded sign-on script may need to be adjusted accordingly.

```
IKJ56700A ENTER USERID -
IKJ56714A ENTER CURRENT PASSWORD FOR XXXX-
ICH70001I XXXX LAST ACCESS AT 13:59:30 ON THURSDAY, JANUARY 15, 2015
IKJ56496I DEFAULT ACCOUNT NUMBERS COULD NOT BE OBTAINED - ENTER ACCOUNT NUMBER
IKJ56481I THE PROCEDURE NAME DBSPROCA IS A DEFAULT NAME - YOU MAY CHANGE IT
IKJ56455I XXXX LOGON IN PROGRESS AT 14:05:08 ON JANUARY 15, 2015
IKJ56951I NO BROADCAST MESSAGES
*****
* APPLICATION DEVELOPER'S CONTROLLED DISTRIBUTION (ADCD) *
*****
* *
```

```
* USER.CLIST(ISPFCL) PRODUCES THIS MESSAGE *
* USER.* DATASETS CONTAIN SYSTEM CUSTOMIZATION BY WP *
* ADCD.* DATASETS CONTAIN SYSTEM CUSTOMIZATION *
* SMP/E DATASETS CAN BE LOCATED FROM 3.4 WITH DSNAME **.CSI *
* *
*****
READY
```

### 3. Write the Telnet sign-on script.

Having just performed a manual sign-on, it is necessary to write a script to automate the process, informed by the sequence of challenges and responses observed during the manual sign-on. In the following commentary, each line of script is explained by a short narrative.

#### Note:

A complete, 'clean' sample script (for potential cut/paste and modification) is included in *Sample Telnet authentication script* [↗](#) (page 39).

```
TRACE ON;
```

The `TRACE ON` statement sends the statements to the log as they are executed by WPS. This is useful for debugging, but it would probably be turned off for production.

```
ECHO ON;
```

The `ECHO ON` statement causes all responses received from the Telnet server to be echoed to the log - again, this is useful for debugging purposes.

```
LOG "NOTE: Signon script entered.";
```

This statement prints a message to the log to indicate that the script is being processed. `LOG` statements can be used liberally within the sign-on script to demonstrate progress.

```
IF signoff THEN GOTO signoff;
```

This line is a standard part of most sign-on scripts. A special variable `signoff` is set if the script is being run as a consequence of the `SIGNOFF` statement rather than the `SIGNON` statement. This allows the one script to cope with both situations. Here, control branches to the `signoff` label if a sign-off is detected.

#### Note:

There is no significance in the name of the label - it is simply conventional to call it `signoff`.

```
WAITFOR "ENTER USERID -", 5 seconds : fail;
TYPE "&USER" ENTER;
```

Here, a `WAITFOR` statement is invoked, to wait for the given line to be received from the Telnet server. The statement causes the script to wait until a line is received containing the given text - in this case, `ENTER USERID -` - somewhere in its contents. If it fails to receive the given prompt within 5 seconds, it will branch to the subsequent `fail` label. If the response is received, processing continues on the next line that simulates the user typing. Here, the response is the user name, which is provided to the sign-on script via a macro variable `&USER`.

```
WAITFOR "ENTER CURRENT PASSWORD", 5 seconds : fail;
TYPE "&PASSWORD" ENTER;
```

The next expected prompt is the password prompt, and again it is provided via a macro variable.

## Note:

it is not necessary to capture and match against the full prompt - the `WAITFOR` command simply looks for a line containing the given text somewhere within it.

```
WAITFOR "ENTER ACCOUNT NUMBER", 5 seconds : fail;
TYPE "WPS" ENTER;
WAITFOR "YOU MAY CHANGE IT", 5 seconds : fail;
TYPE ENTER;
```

Next comes a prompt asking for an `ACCOUNT NUMBER`, to which the response will be installation-dependent. Here, it is `WPS`. Then may come a prompt about a procedure name, to which an empty `ENTER` response might be a satisfactory reply. Depending on the `z/OS` configuration, one or more additional responses may also have to be considered and provided using the same techniques - refer to the exchanges that occurred during the initial manual sign-on to determine if any more such conversations need to be accommodated.

```
WAITFOR "READY", 5 seconds : fail;
LOG 'NOTE: Logged onto z/OS. Starting remote WPS now...';
```

The next prompt to wait for is the `READY` prompt, indicating that authentication is complete and that the `TSO` session is ready to receive commands. This illustrates a feature of the `WAITFOR` statement - multiple lines are received before the `READY` prompt arrives, but these are scanned and ignored. It is not necessary to include a `WAITFOR` statement for every line of output that the sign-on process generates.

```
TYPE "altlib activate application " lf;
WAITFOR 'ENTER Application library', 5 seconds :fail;
TYPE "CLIST" lf;
WAITFOR "ENTER a single dataset", 5 seconds :fail;
TYPE "'WPS.V310.B31754.CLIST'" lf;
WAITFOR "READY", 5 seconds : fail;
type "TSOWPS OPTIONS('DMR WPSCOMPROTOCOL=WPS') TRACE" enter;
```

Here, the commands necessary to launch `WPS` are invoked. The `WPS CLIST` library is temporarily added to the `CLIST` search path using the `ALTLIB` command - the specific library name is dependent on the `WPS` version and build number. Then the `TSOWPS CLIST` is explicitly invoked. This may have already been moved into a user `CLIST` library, in which case the `ALTLIB` command would be unnecessary. To invoke `WPS` as a server for use with **WPS Communicate**, the `DMR` option is required, and while `WPSCOMPROTOCOL` defaults to `WPS` on most installations, it does no harm to be explicit. Finally, the `TRACE` setting is optional and can be omitted if the `CLIST` output is too verbose.

```
WAITFOR "SESSION ESTABLISHED", 5 seconds : fail;
LOG 'NOTE: WPS Communicate conversation established.';
STOP;
```

Having launched the WPS process, the script waits for the output line indicating that the WPS server is running and is waiting for the secondary connection. This line always contains the string `SESSION ESTABLISHED`, and the successful path through the sign-on script should always end with a `WAITFOR` statement. Once this line has been received, the `STOP` statement terminates the sign-on script processing, handing control back to WPS.

```
signoff:
WAITFOR 'READY', 5 seconds : fail;
TYPE 'LOGOFF' ENTER;
WAITFOR "LOGGED OFF", 5 seconds : fail2;
LOG 'NOTE: WPS Communicate conversation terminated.';
STOP;
```

This section contains the lines that are executed when a `SIGNOFF` occurs. The WPS process will have been signalled to terminate, but a pause is necessary to wait for it to end and for the TSO `READY` prompt to be received. Once this has been received, a typed `LOGOFF` command is simulated, waiting for the confirmation message before terminating the script with a `STOP` statement.

```
fail:
LOG "ERROR: Expected prompt not received";
TYPE "LOGOFF" enter;

fail2:
ABORT;
```

The `fail` and `fail2` are labels announcing code that is activated when expected responses are not encountered, including when attempting to log off. They signal failure conditions which, if reached, will need to be investigated further.

#### 4. Test the sign-on script.

Having written a basic sign-on script and saved it to a known location, you should test it by making a connection with it. This requires a simple WPS program such as:

```
filename rlink '<path to signon script>';

%let HOST=zoshost1;
%let USER=XXXX;
%let PASSWORD=XXXX;

signon HOST;
rsubmit;
%PUT &SYSHOSTNAME;
endrsubmit;
signoff;
```

You will need to substitute the `filename rlink` statement with the full path to the sign-on script, and supply the host name, together with a suitable username and password for logging onto the z/OS host. These should be the same as were used earlier during the manual logon.

If all goes well, an error-free WPS log should be created. There will some additional debug output due to the `ECHO` and `TRACE` statements in the sign-on script, but, if the sign-on was successful, such statements can be either commented out - by putting `/* */` around them - or deleted, to make the output less verbose. The `&SYSHOSTNAME` macro variable will be resolved by the remote host and its value written to the local WPS log, demonstrating that the connection and sign-on were successful.

## Sample Telnet authentication script

```
TRACE ON;

ECHO ON;

LOG "NOTE: Signon script entered.";

IF signoff THEN GOTO signoff;

WAITFOR "ENTER USERID -", 5 seconds : fail;
TYPE "&USER" ENTER;

WAITFOR "ENTER CURRENT PASSWORD", 5 seconds : fail;
TYPE "&PASSWORD" ENTER;

WAITFOR "ENTER ACCOUNT NUMBER", 5 seconds : fail;
TYPE "WPS" ENTER;
WAITFOR "YOU MAY CHANGE IT", 5 seconds : fail;
TYPE ENTER;

WAITFOR "READY", 5 seconds : fail;
LOG 'NOTE: Logged onto z/OS... Starting remote WPS now.';

TYPE "altlib activate application " lf;
WAITFOR 'ENTER Application library', 5 seconds :fail;
TYPE "CLIST" lf;
WAITFOR "ENTER a single dataset", 5 seconds :fail;
TYPE "'WPS.V310.B31754.CLIST'" lf;
WAITFOR "READY", 5 seconds : fail;
type "TSOWPS OPTIONS('DMR WPSCOMPROTOCOL=WPS') TRACE " enter;

WAITFOR "SESSION ESTABLISHED", 5 seconds : fail;
LOG 'NOTE: WPS Communicate conversation established.';
STOP;

signoff:
WAITFOR 'READY', 5 seconds : fail;
TYPE 'LOGOFF' ENTER;
WAITFOR "LOGGED OFF", 5 seconds : fail2;
LOG 'NOTE: WPS Communicate conversation terminated.';
STOP;

fail:
LOG "ERROR: Expected prompt not received";
TYPE "LOGOFF" enter;

fail2:
ABORT;
```

# SSH (Secure Shell) from a Windows client

This section covers the use of SSH with both **WPS Communicate** and **WPS Link** to create and maintain the connections between server and client machines.

---

**Note:**

The differences in use between **WPS Communicate** and **WPS Link** are highlighted where appropriate.

---

Before you access a remote host via **WPS Communicate** or **WPS Link**, it is important to ensure that you can access the remote host manually via an external SSH client such as PuTTY. This demonstrates that you can at least connect to the machine using the SSH protocol and that your user ID and password are valid.

---

**Note:**

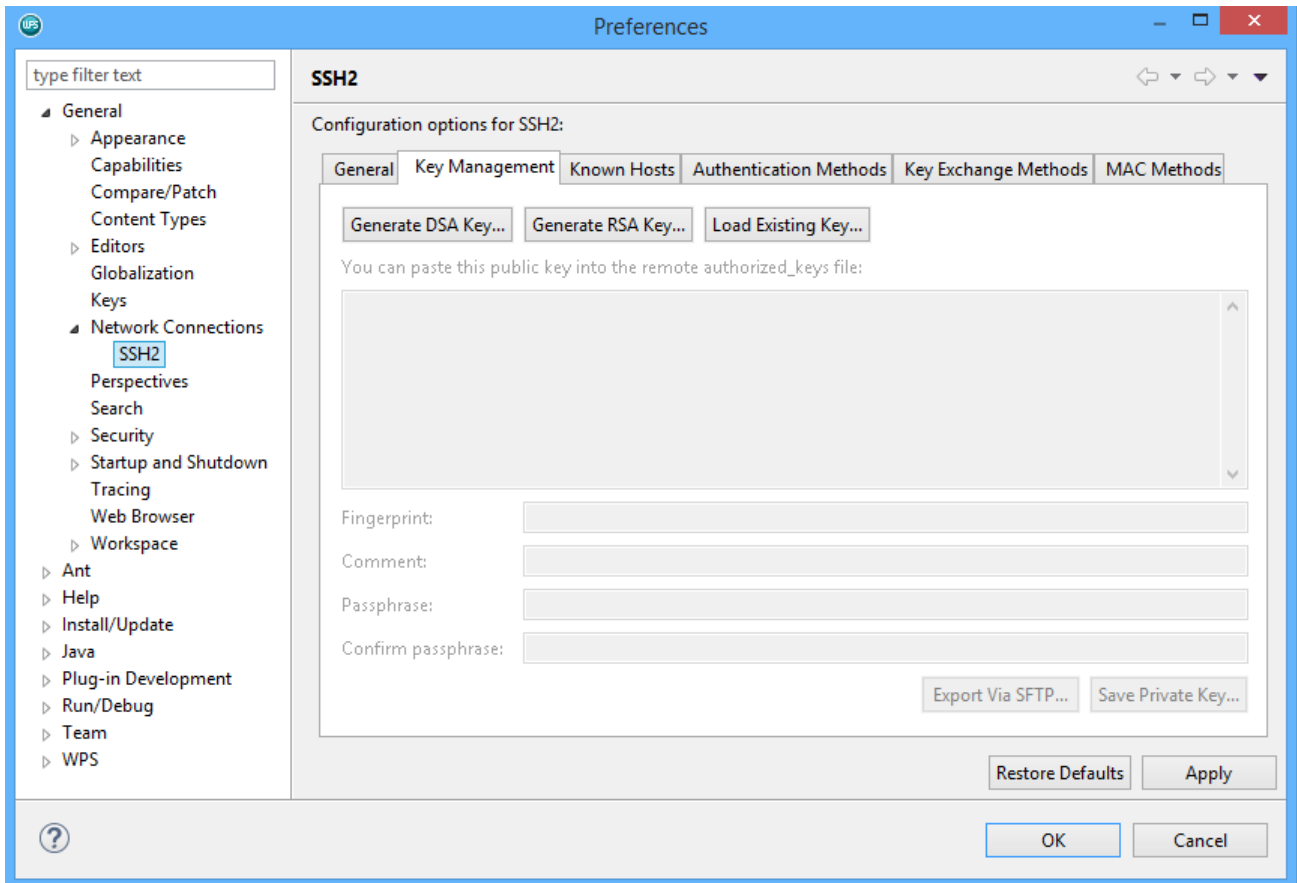
If you intend to use *Public key authentication* [↗](#) (page 47), and keys have not already been generated on the server, then you may also wish to download PuTTYgen (refer to *Key generation using PuTTYgen* [↗](#) (page 48)). If you intend to use public keys with a **passphrase**, and you are using **WPS Communicate**, you will also need to download a keychain agent such as Pageant (refer to *Passphrase authentication using Pageant* [↗](#) (page 60)). The use of such an agent for **passphrases** is not necessary with **WPS Link**, although it may be desirable if you are connecting to multiple servers.

---

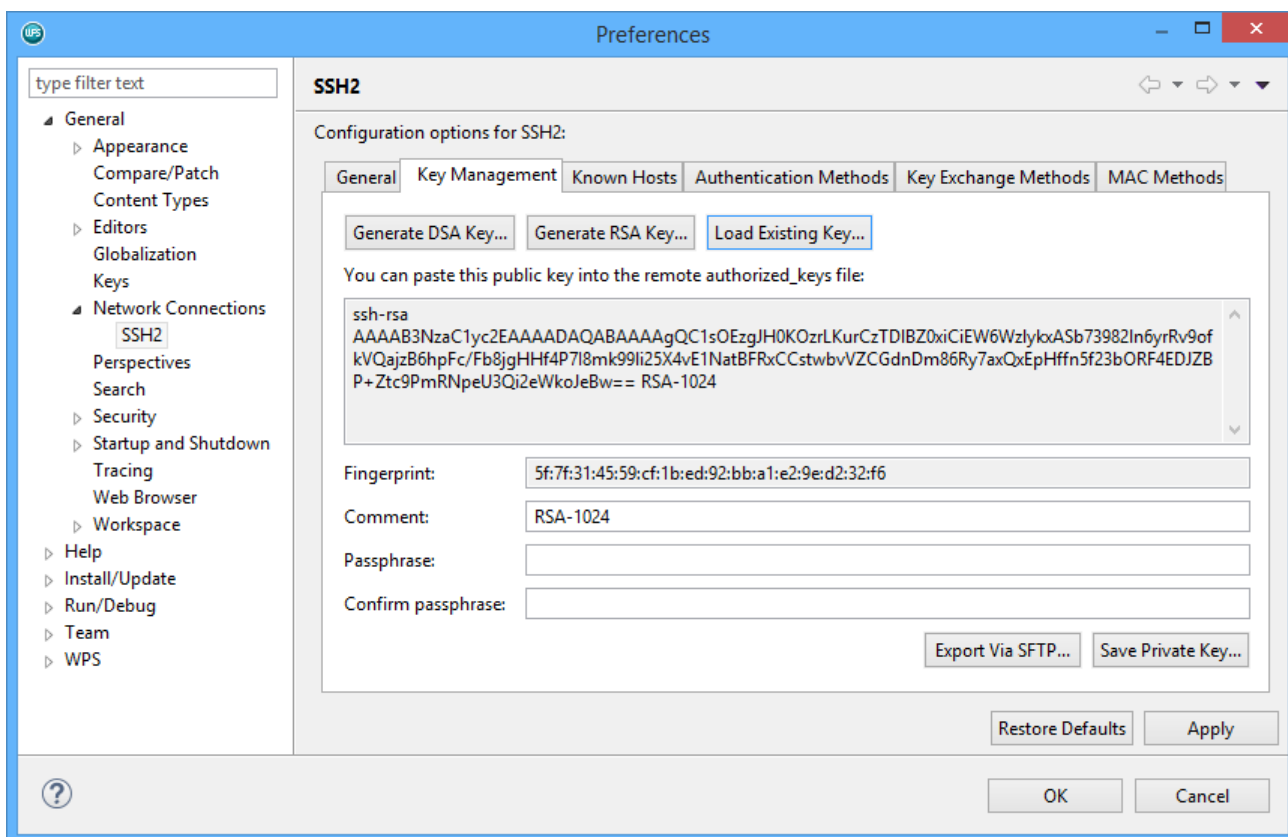
If you are using **WPS Link** and already have a private key, and wish to apply it, then proceed as follows:

1. On the WPS Workbench main menu, select **Window > Preferences** and, in the left-hand pane of the subsequent **Preferences** dialog, expand the **General > Network Connections > SSH2** nodes.
2. Select the **Key Management** tab of the **Preferences** dialog:





3. Click **Load Existing Key....**
4. Browse to the required private key and select it, to display the screen shown in the following example:

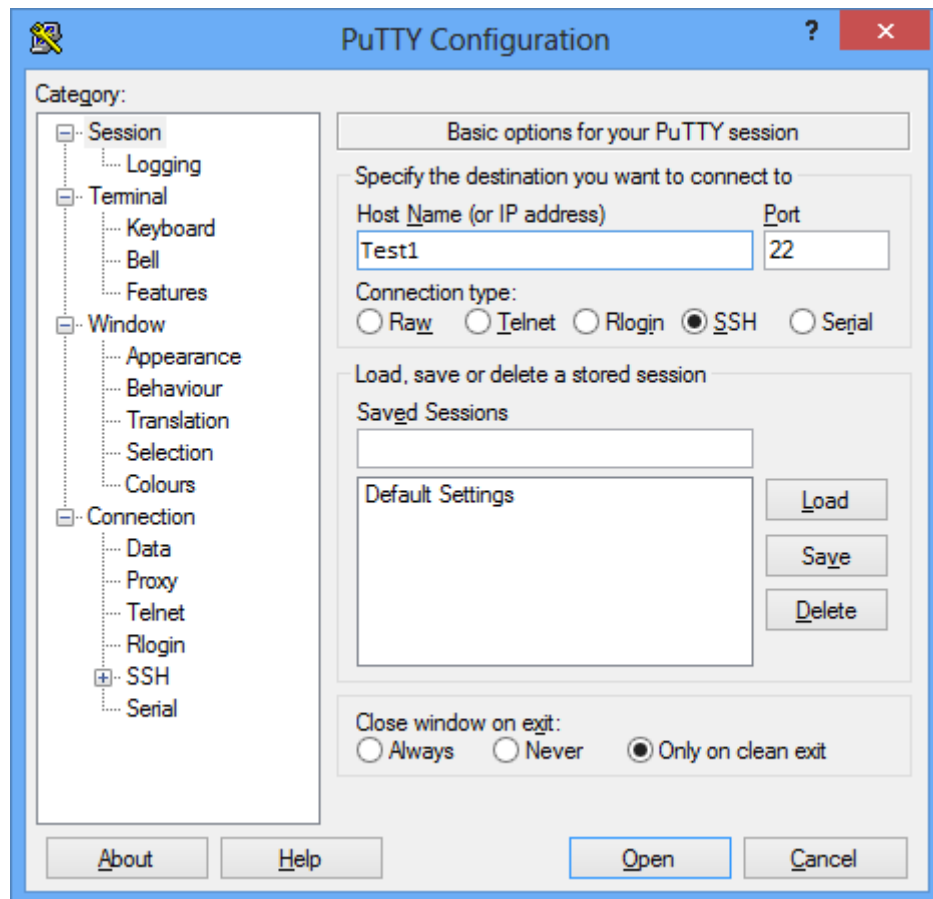


5. If you wish to apply a passphrase to your private key file, complete the **Passphrase** and **Confirm passphrase** fields.
6. Click **OK** to save your changes and dismiss the **Preferences** window.

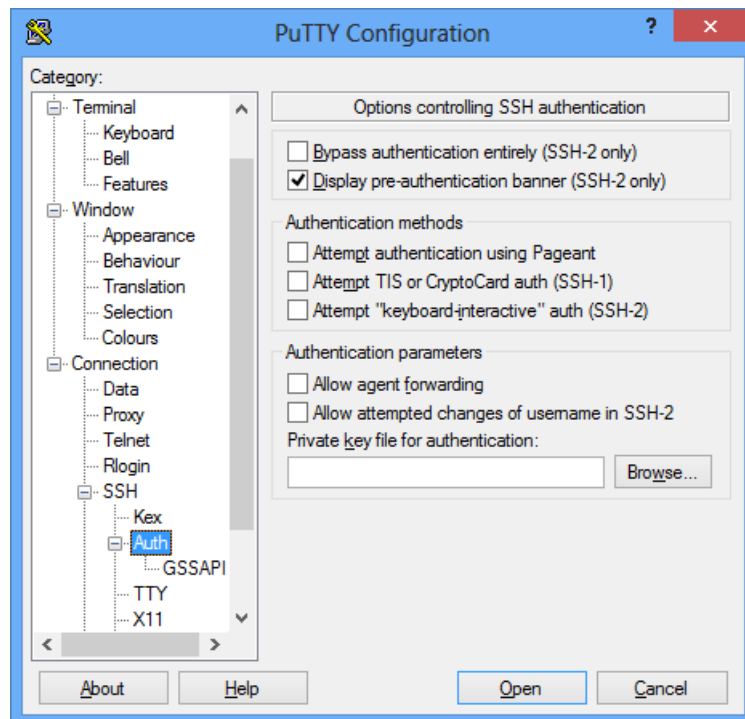
## Password authentication (using PuTTY) and WPS sign-on

Manual SSH sign-on provides the opportunity to perform host key validation. For increased security, WPS performs host key validation during SSH sign-on. However, WPS has no mechanism for interacting with the user to accept new host keys, or to prompt about an apparent change of key. Instead, WPS relies upon host key acceptance having already been performed by an external SSH client, and will validate the host key it receives against the same database as is used by the external SSH client. On Windows clients, WPS will, by default, use the PuTTY host key database stored in the Windows registry, so it is necessary to log onto the remote host using the PuTTY SSH client to validate the host key and add it to the host key database before attempting to make a connection with WPS.

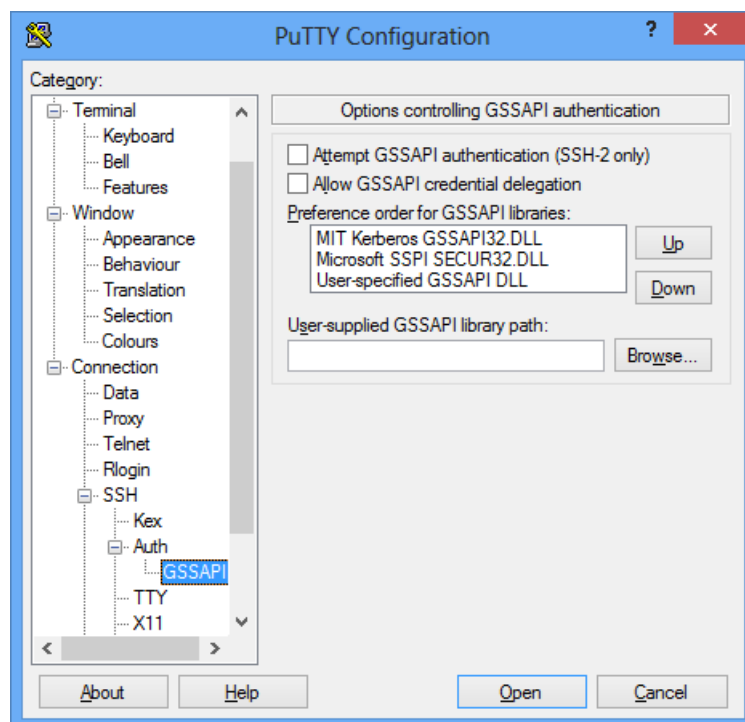
1. Launch the PuTTY client and enter the host name in the main **Host Name (or IP address)** entry field:



2. Expand the **SSH > Auth** configuration page from the category list on the left, and ensure that nothing is selected under **Authentication methods** and that the **Private key file for authentication** field is empty:



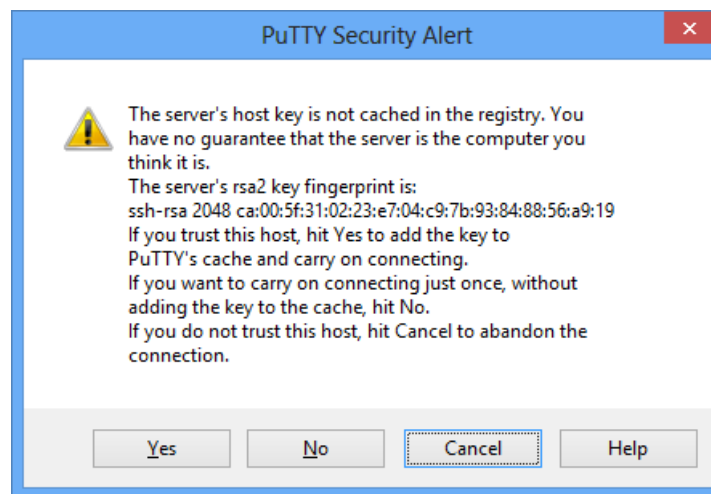
3. Select the **GSSAPI** page and ensure that **Attempt GSSAPI authentication** is not selected:



These checks ensure that only password authentication is being used, and that we have not inadvertently selected some more involved authentication mechanism.

4. Click on **Open** and when prompted, type in your password and press **Enter**.

If this is the first time you have signed on to this particular host, an alert of the following kind will be displayed:



At this point you should confirm that this is indeed the correct fingerprint for the host, and, assuming that it is, click **Yes** to accept the key permanently. This will later allow WPS to perform host key validation using the same cached key. If everything has worked, you will be logged in via a terminal session to your remote host. The host key will have been validated and stored in the Windows registry which is where WPS components will look for it. You can now log out safe in the knowledge that, when you launch WPS, it will be able to access the same remote server, automatically extracting the validated host key from the Windows registry to perform validation.

**Note:**

Do not mistake this host key validation for public key authentication - they are two entirely separate things. Host key validation simply gives you an opportunity to confirm that the host to which you are connecting is, indeed, the host to which you intended to connect.

5. If you are using **WPS Link**, create the required host connection and remote host server through **WPS Workbench**. If you are using **WPS Communicate**, sign onto WPS via the `SIGNON` statement - you need to specify either the `IDENTITYFILE` statement option or the `SSH_IDENTITYFILE` system option, for example:

```
SIGNON <servername> SSH
USERNAME="<username>"
password="<password>"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";

RSUBMIT;
%PUT &SYSHOSTNAME;
ENDRSUBMIT;
SIGNOFF;
```

Alternatively:

```
OPTIONS SSH_IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk";  
SIGNON <servername> SSH  
password="<password>"  
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr";  
  
RSUBMIT;  
%PUT &SYSHOSTNAME;  
ENDRSUBMIT;  
SIGNOFF;
```

**Note:**

You cannot use either **IDENTITYFILE** or **SSH\_IDENTITYFILE** if you are using *Passphrase authentication using Pageant* [↗](#) (page 60).

## Launch command syntax

If you are connecting to a Windows SSH server, a Windows-style launch command path is required. In this example, quotes are required because the path includes spaces:

```
'C:\Program Files\World Programming\WPS\3\bin\wps' -dmr
```

If you are connecting to a UNIX/Linux server, the path might be:

```
/home/installs/wps-3.2/bin/wps -dmr
```

## Enhancement of username/password authentication

So far, the examples of signing-on via WPS using SSH have relied upon a `SIGNON` statement that directly contains the user name and password. This is not ideal, as it means storing these confidential details in a source file.

Currently, WPS does not support prompting for credentials. However, the password can be obfuscated using `PROC PWENCODE`.

**Note:**

This is not a strong encryption method. WPS currently supports the BASE64 mechanism for obfuscation.

To use `PROC PWENCODE` with a `SIGNON` statement:

1. Run the following program to encode your password:

```
proc pwencode in="<password>" out=log;  
run;
```

This would produce something similar to the following in the log:

```
43      proc pwencode in=<password> out=log;
44      run;
{sas001}dG9wc2VjcmV0
NOTE: Procedure pwencode step took :
      real time : 0.004
      cpu time  : 0.000
```

**Note:**

You would run this program as an offline task.

2. Copy and paste the encoded password into your SIGNON program:

```
SIGNON docserver SSH
username="<username>"
password="{sas001}dG9wc2VjcmV0"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr";
```

## Public key authentication

This method is more secure than using a simple password, and is sometimes called *password-less* authentication.

With SSH, authentication using keys not only improves security in general, but also, in the case of **WPS Communicate**, it avoids having user names and passwords committed to source code (even in obfuscated or encrypted form).

This authentication method relies upon a cryptographic key-pair, where the private key resides on (and never leaves) the client machine, and the public key is installed on the SSH server to which the client needs to connect, or, in the case of Windows, on **Bitvise SSH Server** (from WPS 3.2 onwards). The SSH protocol uses the key-pair to establish the identity of the client and perform the authentication.

Two methods are described by which the keys can be generated on a Windows client:

- *Key generation using PuTTYgen* [↗](#) (page 48)
- *Key generation using WPS Workbench* [↗](#) (page 50)

Following the generation of the public keys, they should be placed on the remote server in accordance with *Deploying public keys on the remote SSH server* [↗](#) (page 53), or, in the case of a Windows server, *Deploying public keys on Bitvise SSH Server* [↗](#) (page 55).

If you wish to connect to multiple servers, without having to remember or enter your password for each system, then you should also use *Passphrase authentication using Pageant* [↗](#) (page 60).

The validity of the key-pairs should then be checked in accordance with *Remote host access verification (using PuTTY) and WPS sign-on* [↗](#) (page 58).

**Note:**

Public key authentication can be used with both **WPS Communicate** and **WPS Link**. However, for **WPS Communicate**, if you are not using a keychain agent such as Pageant, there cannot be a **passphrase** on the private key file, as there is currently no interactive mechanism to prompt for it during WPS authentication.

**Note:**

You need to ensure that public key authentication is not disabled on the client machine.

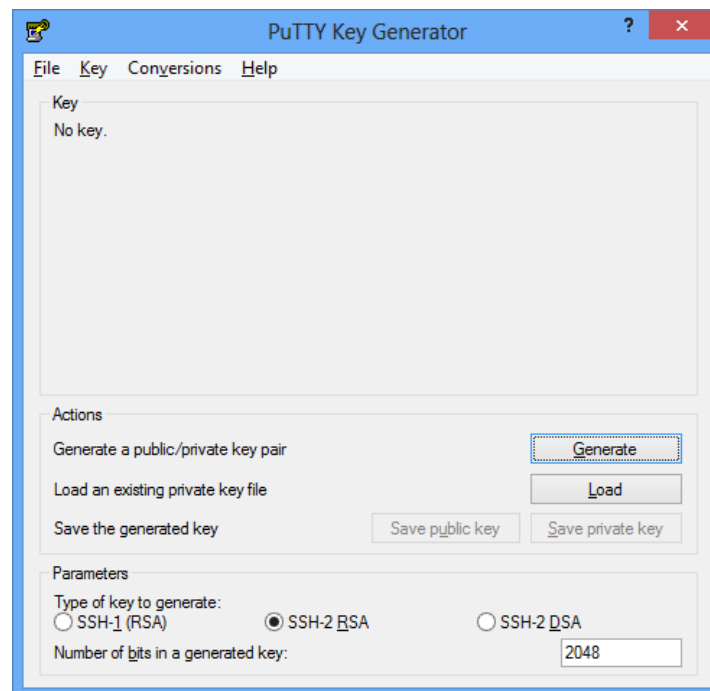
## Key generation using PuTTYgen

To generate a key-pair, which is the combination of the private key and the public key for asymmetric encryption, proceed as follows:

**Note:**

You should be aware that, as an alternative, you can also use WPS Workbench to generate key-pairs (refer to *Key generation using WPS Workbench* [↗](#) (page 50)).

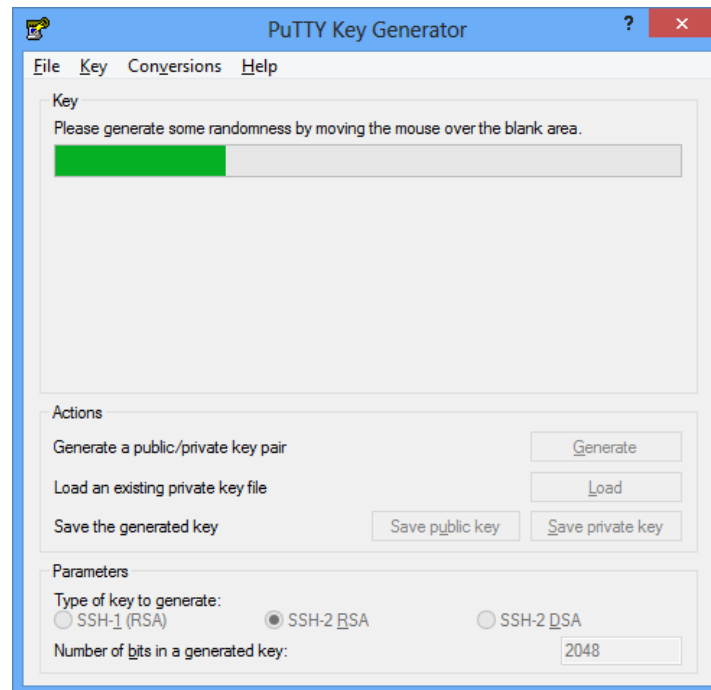
1. Launch the PuTTYgen tool (available via the same sources as PuTTY).

**Important:**

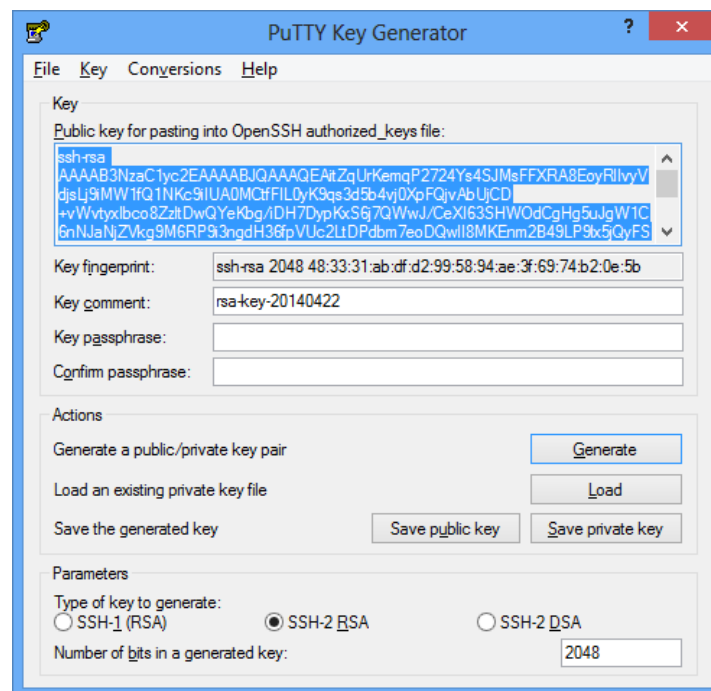
It is recommended that you use the parameter **SSH-2 RSA** with a minimum key length of **2048**.



2. Click the **Generate** button and move the mouse within the indicated area to generate some randomness - this will act as a seed for your key-pair:



3. The system generates a key-pair:



4. If you wish to apply a passphrase to your private key file, complete the **Key passphrase** and **Confirm passphrase** fields. You will need to do this if you are going to be using *Passphrase authentication using Pageant* [\(page 60\)](#).

**Note:**

If you are using **WPS Communicate**, do not enter a **passphrase** unless you are going to be using *Passphrase authentication using Pageant* [↗](#) (page 60). If you are using **WPS Link**, you can enter a **passphrase** and use it either with or without *Passphrase authentication using Pageant* [↗](#) (page 60).

5. Click **Save private key**. If you did not enter a **passphrase**, you will be asked to confirm that you wish to save the key without a **passphrase**. The resultant file is in PuTTY's native format (\* .PPK), and, when you are prompted to save the file to a folder, you should ensure that it is stored in the .ssh folder in your user profile.

**Note:**

Ensure that the permissions on your private key file are such that only you can read it. This file is essentially your password, so it is important that no-one else can access the file.

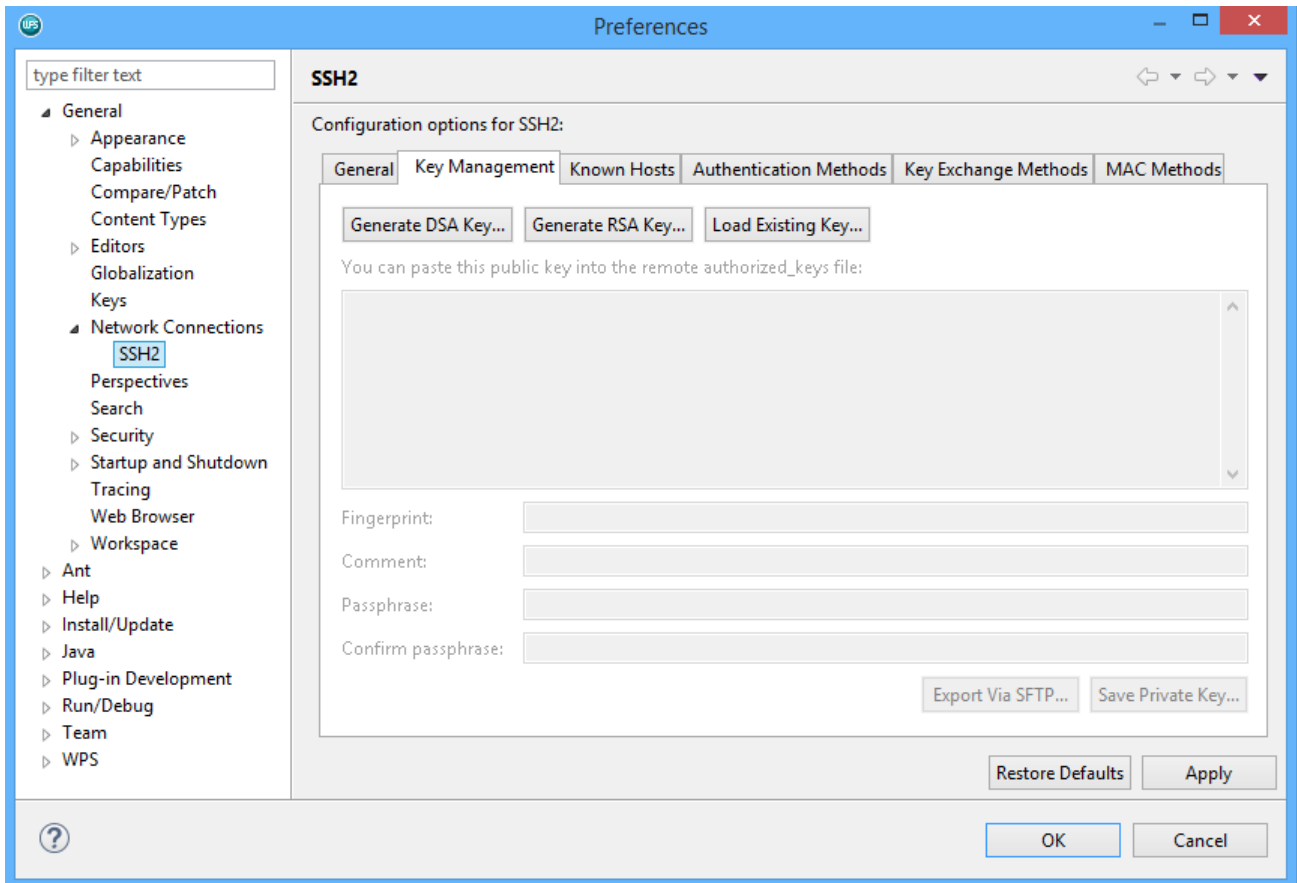
6. The public key that is highlighted in step 3 [↗](#) (page 49) contains the information needed to allow a user to verify that another party is in possession of the corresponding private key. The public key does not need to be kept secure, but you should save it by either copying it to your pasteboard and pasting it to a plain text file, or selecting **Save public key** in order to save it to the .ssh folder in your user profile. You should then proceed as in *Deploying public keys on the remote SSH server* [↗](#) (page 53) or *Deploying public keys on Bitvise SSH Server* [↗](#) (page 55).

**Note:**

If you are going to use **WPS Link** in conjunction with **WPS Communicate**, then, in order to avoid the need for two separate key-pairs, you should have a consistent strategy - that is to say, either avoid a **passphrase** in both cases, or else create a single **passphrase** and associate it with a single key-pair via *Passphrase authentication using Pageant* [↗](#) (page 60).

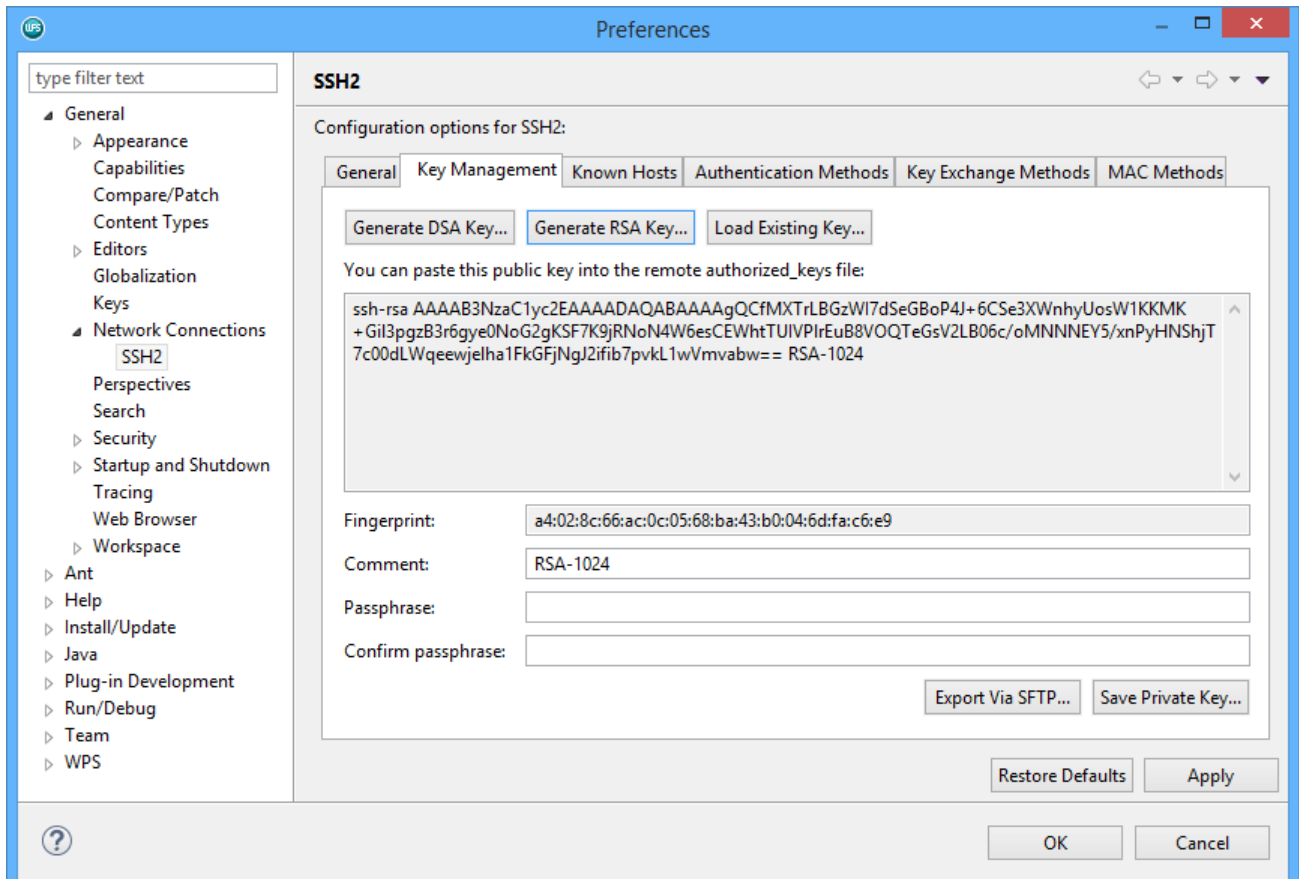
## Key generation using WPS Workbench

1. On the WPS Workbench main menu, select **Window > Preferences** and, in the left-hand pane of the subsequent **Preferences** dialog, expand the **General > Network Connections > SSH2** nodes.
2. Select the **Key Management** tab of the **Preferences** dialog:



### 3. Click **Generate RSA Key....**

A key pair is generated - the public key is displayed in a text box in the centre of the dialog, for example:



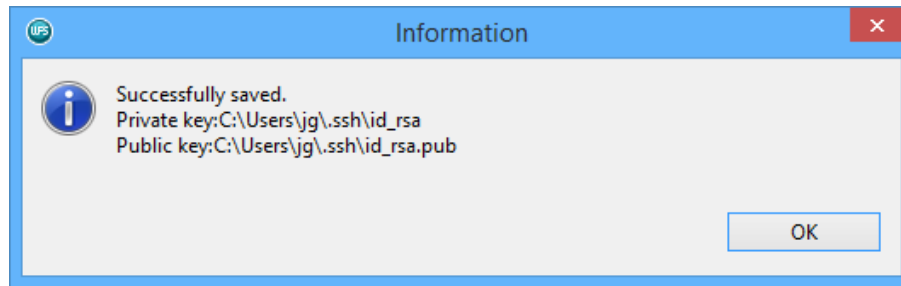
- If you wish to apply a passphrase to your private key file, complete the **Passphrase** and **Confirm passphrase** fields. You will need to do this if you are going to be using *Passphrase authentication using Pageant* [↗](#) (page 60).

#### Note:

If you are using **WPS Communicate**, do not enter a **passphrase** unless you are going to be using *Passphrase authentication using Pageant* [↗](#) (page 60). If you are using **WPS Link**, you can enter a **passphrase** and use it either with or without *Passphrase authentication using Pageant* [↗](#) (page 60).

- Click **Save Private Key**. If you did not enter a **passphrase**, you will be asked to confirm that you wish to save the key without a **passphrase**. When you are prompted to save the resultant key file to a folder, you should ensure that it is stored in the `.ssh` folder in your user profile. If you do not wish to use the default name of `id_rsa`, give the file a more meaningful name.

WPS Workbench displays an information dialog confirming that it has saved your private key file, together with the corresponding public key file. It gives the public key file the same prefix as your private key file, but appends `.pub` to it, for example:

**Note:**

Ensure that the permissions on your private key file are such that only you can read it. This file is essentially your password, so it is important that no-one else can access the file.

6. Click **OK** to dismiss the **Information** dialog.
7. Click **OK** to save your changes and dismiss the **Preferences** window.
8. The public key that is both displayed on screen (for copying and pasting if required), and saved to a file, contains the information needed to allow a user to verify that another party is in possession of the corresponding private key. You should then proceed as in *Deploying public keys on the remote SSH server* [↗](#) (page 53) or *Deploying public keys on Bitvise SSH Server* [↗](#) (page 55).

**Note:**

If you are going to use **WPS Link** in conjunction with **WPS Communicate**, then, in order to avoid the need for two separate key-pairs, you should have a consistent strategy - that is to say, either avoid a **passphrase** in both cases, or else create a single **passphrase** and associate it with a single key-pair via *Passphrase authentication using Pageant* [↗](#) (page 60).

## Deploying public keys on the remote SSH server

1. Log into the remote machine.
2. Once logged in, you must configure the server to accept your public key for authentication, so change into the `.ssh` directory and open the file `authorized_keys`.

If this is the first public key to be put into the file, then you may need to create the directory and file first, by, for example, running the following commands:

```
mkdir -p .ssh
touch ~/.ssh/authorized_keys
```

3. Set the right permissions, for example:

```
chmod 600 ~/.ssh/authorized_keys
```

**Note:**

You also need to ensure that your `$HOME` directory and `.ssh` directory have the permissions that are appropriate to both the server and your particular operation.

4. Now you can add the public key to the `authorized_keys` file, as in the following example:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

If you currently have password-based SSH access configured to your server, and you have the `ssh-copy-id` utility installed, then you can simply transfer your public key by typing:

```
ssh-copy-id username@remote_host
```

You will then be prompted for the user account's password on the remote system. After typing in the password, the contents of your `~/.ssh/id_rsa.pub` key will be appended to the end of the user account's `~/.ssh/authorized_keys` file. You can then log into that account without a password:

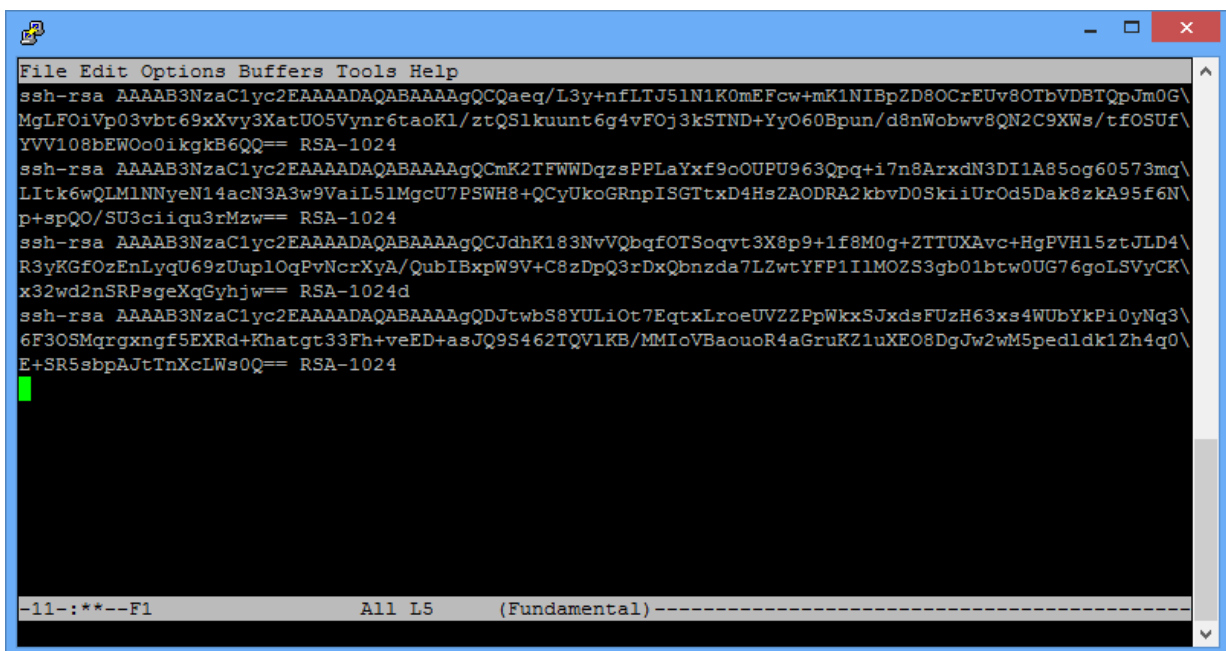
```
ssh username@remote_host
```

Alternatively, you can copy and paste the public key from PuTTYgen or **WPS Workbench** into the `authorized_keys` file, ensuring that it ends up on a single line.

5. Verify the contents of `~/.ssh/authorized_keys` to ensure that your public key was added properly, by entering the following on the command line:

```
more ~/.ssh/authorized_keys
```

The contents of a typical `~/.ssh/authorized_keys` file might resemble:



```
File Edit Options Buffers Tools Help
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQCQaeq/L3y+nfLTJ5lN1K0mEFcw+mK1NIBpZD8OCrEUv80TbVDBTQpJm0G\
MgLF0iVp03vbt69xXvy3XatU05Vynr6taoKl/ztQS1kuunt6g4vFOj3kSTND+YyO60Bpun/d8nWobwv8QN2C9XWs/tfOSUf\
YVV108bEW0o0ikgkB6QQ== RSA-1024
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQCmK2TFWWDqzsPPLaYxf9oOUPU963Qpq+i7n8ArxdN3DI1A85og60573mq\
LItk6wQLM1NNyeN14acN3A3w9VaiL5lMgcU7PSNH8+QCyUkoGRnpISGTtxD4HsZAODRA2kbvD0SkiUUrOd5Dak8zkA95f6N\
p+spQO/SU3ciiqu3rMzw== RSA-1024
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQCJdhK183NvVQbqfOTSoqvt3X8p9+1f8M0g+ZTTUXAvc+HgFVH15ztJLD4\
R3yKGfOzEnLyqU69zUuplOqPvNcrXyA/QubIBxpW9V+C8zDpQ3rDxQbnzda7LZwtYFP1IIM0ZS3gb01btw0UG76goLSVyCK\
x32wd2nSRPsgexGyjhjw== RSA-1024d
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQDJtwbS8YULiOt7EqtXLroeUVZ2PpWkxSJxdsFUzH63xs4WUbYkPi0yNq3\
6F30SMgrgxngf5EXRd+Khatgt33Fh+veED+asJQ9S462TQV1KB/MMIoVBAouoR4aGruKZ1uXE08DgJw2wM5pedldk1Zh4q0\
E+SR5sbpAJtTnXcLWs0Q== RSA-1024

-11-:***--F1 All L5 (Fundamental)-----
```

**Note:**

If you look carefully, you can see that the above file contains four public keys - each begins with `ssh-rsa` and ends with a phrase similar to `RSA-1024`.

## Deploying public keys on Bitvise SSH Server

If you are new to Bitvise SSH Server (refer to the Bitvise documentation [🔗](#) for specific configuration details), we highly recommend that you first make sure that you can establish a working SSH connection before you change any settings on the server. If you cannot connect to the SSH server using its default configuration, this is most likely due to a network or firewall problem that you will need to resolve before you are able to connect. In its default configuration, Bitvise SSH Server accepts connections on the often-used port number for SSH servers, 22. This is the only port that you need to open in your firewall in order to connect to the SSH server. If you use port forwarding to tunnel other applications through SSH, you should **not** open any additional ports for the tunnelled connections. All tunnelled connections are forwarded through the SSH session, established through port 22.

1. When connecting to Bitvise SSH Server with an SSH client for the first time, log in with the username and password of a Windows account that exists on the machine where the SSH server is running. To log into a Windows domain account, specify it in the `domain\account` format.

You can use any SSH client to log into Bitvise SSH Server, as long as it supports SSH protocol version 2.

2. Having ensured that the public key has been saved to a file, transfer it to the machine where Bitvise SSH Server is installed, or to the machine from which you manage the SSH Server remotely using Bitvise SSH Client.
3. Open the **SSH Server Control Panel**, and then, to import the public key into the SSH user's account settings, use either **Open easy settings**:

The screenshot displays the Bitvise SSH Server Control Panel. The main window is titled 'Bitvise SSH Server Easy Settings' and has three tabs: '1. Server settings', '2. Windows accounts', and '3. Virtual accounts'. The '2. Windows accounts' tab is active, showing 'Simplified Windows accounts' settings. A sidebar on the left contains links like 'Server m', 'Bitvise SSH', 'Host key', 'Settings', and 'Password cache'. The 'Settings' section has a link 'Open easy settings' labeled with a '1'. The 'Public keys' window is open in the foreground, showing a table of imported public keys. The 'Import' button is labeled with a '4'.

**Bitvise SSH Server Easy Settings**

1. Server settings | 2. Windows accounts | 3. Virtual accounts

**Simplified Windows accounts**

These settings control SSH login rights and permissions for local accounts that already exist in Windows. If your server also control login rights and permissions for domain accounts.

To create or manage local Windows accounts, use C

If using domain accounts, don't fo

Allow login to any Windows

Windows account type | Win

Local account

Add Copy Edit

Back Next

**New entry in Simplified Windows accounts**

Expand / Collapse all help

Windows account type | Local account

Windows account domain

Windows account name | test

Login allowed | ☒

Public keys | 3 | 0 keys

Allow file transfer | ☒

Allow terminal | ☒

Allow port forwarding | ☒

Virtual filesystem layout | Allow full access

**Public keys**

You have imported the following public keys:

Algorithm	Size	MD5 Fingerprint	Bubble Babble	Insert time	Comment

4 Import Remove Remove all

Export Edit Close

or **Edit advanced settings**:



[illegible]

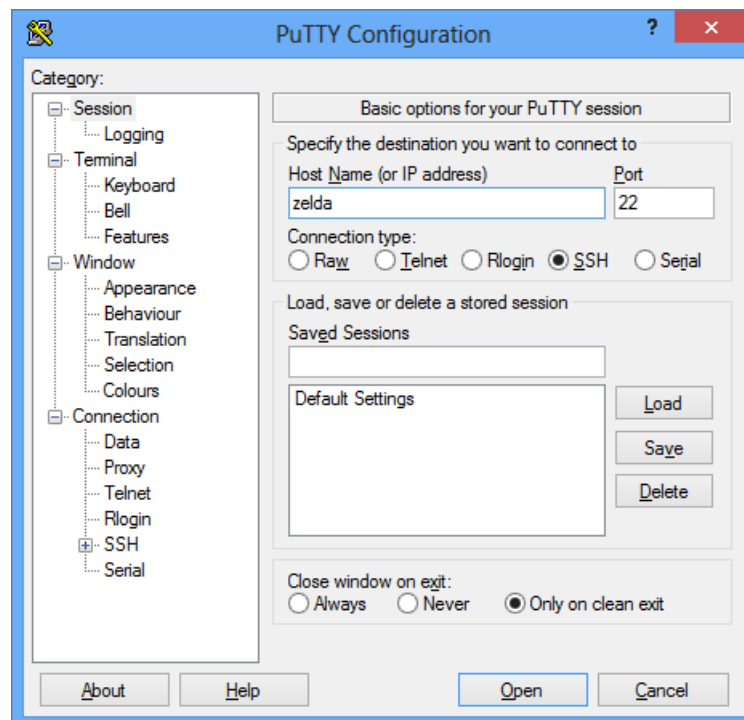
**Note:**

For Windows accounts, Bitvise SSH Server also supports synchronisation with `~/.ssh/authorized_keys`, provided that this feature is enabled in **Advanced SSH Server settings**, under **Access control**. If this feature is enabled, Bitvise SSH Server will check for the existence of the `authorized_keys` file when the user logs out. If the file exists, Bitvise SSH Server will replace all the public keys configured for the user with the keys found in this file.

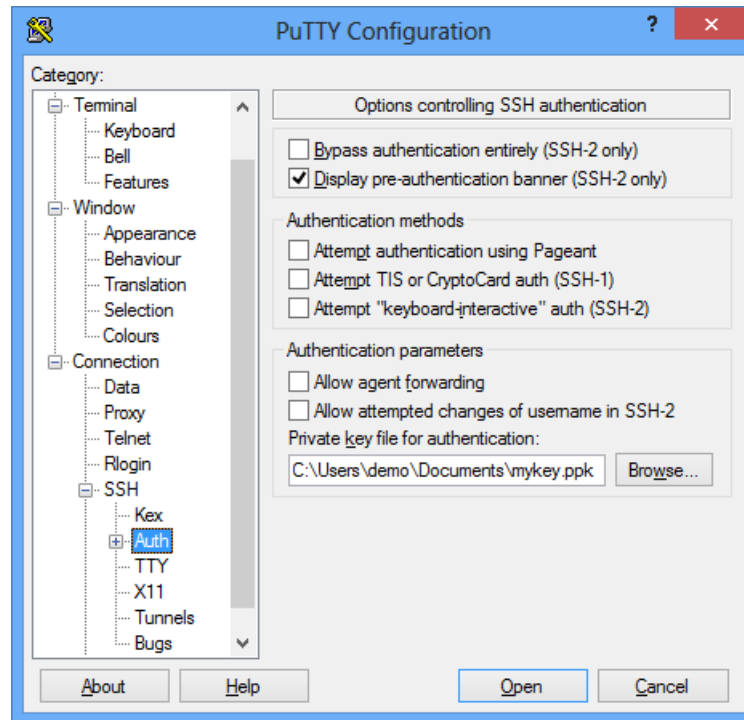
## Remote host access verification (using PuTTY) and WPS sign-on

1. Verify authentication using PuTTY, as follows.

a. Launch the PuTTY client and enter the host name in the **Host Name** field:



b. Select the **SSH > Auth** configuration page from the category list on the left and ensure that nothing is checked under **Authentication methods** (unless you are verifying *Passphrase authentication using Pageant* [↗](#) (page 60), in which case **Attempt authentication using Pageant** needs to be selected).



- c. In the **Private key file for authentication** field, enter the name of the private key file you generated via either *Key generation using PuTTYgen* (page 48) or *Key generation using WPS Workbench* (page 50).
- d. Click **Open** - you will be authenticated via your key-pair combination and a console window will open. The window displays a notification regarding the authentication method.

#### Note:

If you are prompted for a password, then public key authentication has failed.

2. If public key authentication is successful, then, if you are using **WPS Link**, create the required host connection and remote host server through **WPS Workbench**. If you are using **WPS Communicate**, sign onto WPS using your private key, via the `SIGNON` statement, for which you need to specify either the **IDENTITYFILE** statement option or the **SSH\_IDENTITYFILE** system option, for example:

```
SIGNON <servername> SSH
USERNAME="<username>"
IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";
```

Alternatively:

```
OPTIONS SSH_IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk";
SIGNON <servername> SSH
username="<username>"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";
```

#### Note:

You cannot use either **IDENTITYFILE** or **SSH\_IDENTITYFILE** if you are using *Passphrase authentication using Pageant* (page 60).

## Passphrase authentication using Pageant

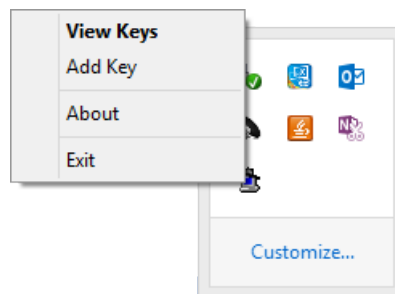
**WPS Communicate** does not support the reading of private key files that have been saved in encrypted form with a **passphrase**. However, it is still possible to use private key-pairs of this form if you use a keychain agent. We will assume that you are using Pageant on Windows.

A keychain agent is a utility that runs on the client machine and stores the decrypted public key file. The SSH client (or WPS when performing an SSH sign-on) contacts the agent for a public key to use when connecting to a given host. Although a password is still required to decrypt the private key, this is only required once - when the keychain agent first opens the private key.

This procedure assumes that you have already generated a key-pair with a **passphrase**, and deployed the public key on the remote SSH server (for UNIX/Linux), or on Bitwise SSH Server (for Windows).

Proceed as follows:

1. Run the Pageant tool, right-click the **system tray** icon and choose **View Keys** or **Add Keys** to add your private key file.



At this point you will be prompted for the **passphrase** for the private key file.

2. Pageant opens and decrypts it, allowing clients (such as the normal PuTTY program or WPS) to request the loaded identity list for authentication.

---

**Note:**

WPS automatically detects whether or not Pageant is running.

---

# SSH (Secure Shell) from a UNIX client

Before you access a remote host via **WPS Communicate** or **WPS Link**, it is important to ensure that you can access the remote host manually via SSH. This demonstrates that you can at least connect to the machine using the SSH protocol and that your user ID and password are valid.

**Note:**

If you intend to use *Public key authentication* [↗](#) (page 62), and keys have not already been generated on the server, then you may wish to use **ssh-keygen** (refer to *Key generation using ssh-keygen* [↗](#) (page 63)). If you intend to use public keys with a **passphrase**, and you are using **WPS Communicate**, you will also need to download a keychain agent such as **ssh-agent** (refer to *Passphrase authentication using ssh-agent* [↗](#) (page 69)). The use of such an agent for **passphrases** is not necessary with **WPS Link**, although it may be desirable if you are connecting to multiple servers.

## Password authentication and WPS sign-on

WPS will use the OpenSSH host key database stored in the `~/.ssh/known_hosts` file by default. It is necessary to log onto the remote host using the OpenSSH client's command line to validate and accept the host key, and ensure that it is added to the `known_hosts` file before attempting to make a connection with WPS. With OpenSSH, it is also possible for a system administrator to add keys manually to the `/etc/ssh/ssh_known_hosts` file, in preference to the `~/.ssh/known_hosts` file.

To sign on via SSH to a remote host, and ensure that password authentication is employed:

1. Issue the following command:

```
ssh -o PreferredAuthentications=password <hostname>
```

2. Verify that you receive a password prompt:

```
<user>@<hostname>'s password:
```

**Note:**

It is important to ensure that password authentication is being used, and that the host is not, for example, configured to accept only public key sign-on.

3. If you are using **WPS Link**, create the required host connection and remote host server through **WPS Workbench**. If you are using **WPS Communicate**, sign onto WPS via the `SIGNON` statement - you need to specify either the **IDENTITYFILE** statement option or the **SSH\_IDENTITYFILE** system option, for example:

```
SIGNON <servername> SSH
USERNAME="<username>"
password="<password>"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";

RSubmit;
%PUT &SYSHOSTNAME;
ENDRSubmit;
SIGNOFF;
```

Alternatively:

```
OPTIONS SSH_IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk";
SIGNON <servername> SSH
password="<password>"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";

RSubmit;
%PUT &SYSHOSTNAME;
ENDRSubmit;
SIGNOFF;
```

---

**Note:**

You cannot use either **IDENTITYFILE** or **SSH\_IDENTITYFILE** if you are using *Passphrase authentication using ssh-agent* [↗](#) (page 69).

---

## Public key authentication

The method described by which the keys can be generated on a UNIX client, is *Key generation using ssh-keygen* [↗](#) (page 63).

Following the generation of the public keys, they should be placed on the remote server in accordance with *Deploying public keys on the remote SSH server* [↗](#) (page 53), or, in the case of a Windows server, *Deploying public keys on Bitvise SSH Server* [↗](#) (page 55).

If you wish to connect to multiple servers, without having to remember or enter your password for each system, then you should also use *Passphrase authentication using ssh-agent* [↗](#) (page 69).

The validity of the key-pairs should then be checked in accordance with *Remote host access verification and WPS sign-on* [↗](#) (page 68).

---

**Note:**

Public key authentication can be used with both **WPS Communicate** and **WPS Link**. However, for **WPS Communicate**, if you are not using a keychain agent such as **ssh-agent**, there cannot be a **passphrase** on the private key file, as there is currently no interactive mechanism to prompt for it during WPS authentication.

---

**Note:**

You need to ensure that public key authentication is not disabled on the client machine.

## Key generation using ssh-keygen

The **ssh-keygen** program allows you to create RSA keys for use by SSH protocol version 2. The type of key to be generated is specified by the `-t` option. If invoked without any arguments, **ssh-keygen** will generate an RSA key for use in SSH protocol 2 connections.

1. Generate the key-pair. In the following example, we have logged onto `hostA` as `wplusr`:

```
ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/wplusr/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/wplusr/.ssh/id_rsa.
Your public key has been saved in /home/wplusr/.ssh/id_rsa.pub.
The key fingerprint is:
f6:61:a8:27:35:cf:4c:6d:13:22:70:cf:4c:c8:a0:23 wplusr@hostA
```

**Note:**

Do not enter a **passphrase** if you are using **WPS Communicate**. You need to use a keychain agent for this (refer to *Passphrase authentication using ssh-agent* [↗](#) (page 69)). In the above example, the private key was saved in `.ssh/id_rsa` (this file is read-only and only for you, and no-one else must see the contents of that file, as it is used to decrypt all correspondence encrypted with the public key), and the public key in `.ssh/id_rsa.pub`. This is the file that needs to be added to the `~/.ssh/authorized_keys` file on the remote machine.

2. To deploy the public key, proceed as in *Deploying public keys on the remote SSH server* [↗](#) (page 53) or *Deploying public keys on Bitvise SSH Server* [↗](#) (page 55).

## Deploying public keys on the remote SSH server

1. Log into the remote machine.
2. Once logged in, you must configure the server to accept your public key for authentication, so change into the `.ssh` directory and open the file `authorized_keys`.

If this is the first public key to be put into the file, then you may need to create the directory and file first, by, for example, running the following commands:

```
mkdir -p .ssh
touch ~/.ssh/authorized_keys
```

3. Set the right permissions, for example:

```
chmod 600 ~/.ssh/authorized_keys
```

**Note:**

You also need to ensure that your `$HOME` directory and `.ssh` directory have the permissions that are appropriate to both the server and your particular operation.

4. Now you can add the public key to the `authorized_keys` file, as in the following example:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

If you currently have password-based SSH access configured to your server, and you have the `ssh-copy-id` utility installed, then you can simply transfer your public key by typing:

```
ssh-copy-id username@remote_host
```

You will then be prompted for the user account's password on the remote system. After typing in the password, the contents of your `~/.ssh/id_rsa.pub` key will be appended to the end of the user account's `~/.ssh/authorized_keys` file. You can then log into that account without a password:

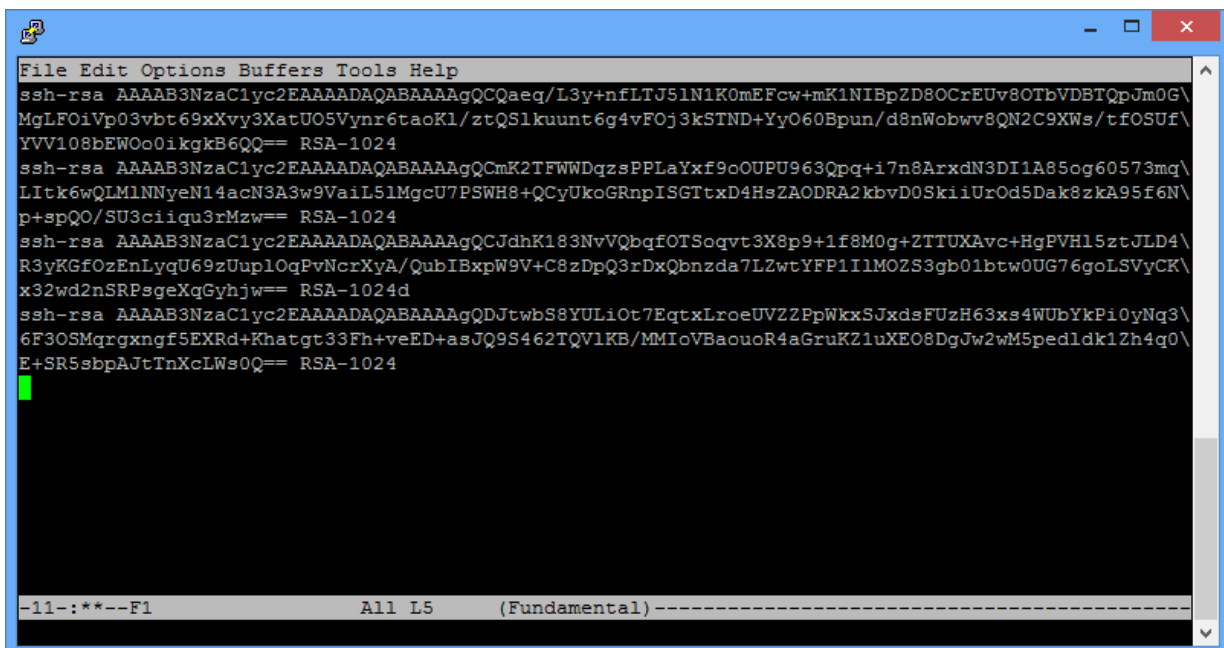
```
ssh username@remote_host
```

Alternatively, you can copy and paste the public key from PuTTYgen or **WPS Workbench** into the `authorized_keys` file, ensuring that it ends up on a single line.

5. Verify the contents of `~/.ssh/authorized_keys` to ensure that your public key was added properly, by entering the following on the command line:

```
more ~/.ssh/authorized_keys
```

The contents of a typical `~/.ssh/authorized_keys` file might resemble:



```
File Edit Options Buffers Tools Help
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCQaeq/L3y+nflTJ51N1K0mEFcw+mK1NIBpZD8OCrEUv8OTbVDBTQpJm0G\
MgLFOiVp03vbt69xXvy3XatU05Vynr6taoKl/zTQSlkuunt6g4vFOj3kSTND+YyO60Bpun/d8nWobwv8QN2C9XWs/tfOSUf\
YVV108bEW0o0ikgkB6QQ== RSA-1024
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCmK2TFWWDqzsPPLaYxf9oOUPU963Qpq+i7n8ArxdN3DI1A85og60573mq\
LItk6wQLM1NNyeN14acN3A3w9VaiL51MgcU7PSWH8+QCyUkoGRnpISGtTxD4HsZAODRA2kbvD0SkiiUrOd5Dak8zkA95f6N\
p+spQO/SU3ciiqu3rMzw== RSA-1024
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCJdhK183NvVQbqfOTSoqvt3X8p9+1f8M0g+2TTUXAvc+HgPVH15ztJLD4\
R3yKGfOzEnLyqU69zUuplOqPvNcrXyA/QubIBxpW9V+C8zDpQ3rDxQbnzda7LZwtYFP1I1MOZS3gb01btw0UG76goLSVyCK\
x32wd2nSRPsgcXgGyhjw== RSA-1024d
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDJtwbS8YULiOt7EqtXLroeUVZ2PpWkxSJxdsFUzH63xs4WUbykPi0yNq3\
6F3OSMqrgxngf5EXRd+Khatgt33Fh+veED+asJQ9S462TQV1KB/MMIoVBaouoR4aGruKZ1uXEO8DgJw2wM5pedldk1Zh4q0\
E+SR5sbpAJtTnXcLWs0Q== RSA-1024
-11-:***-F1          All L5      (Fundamental)-----
```



**Note:**

If you look carefully, you can see that the above file contains four public keys - each begins with `ssh-rsa` and ends with a phrase similar to `RSA-1024`.

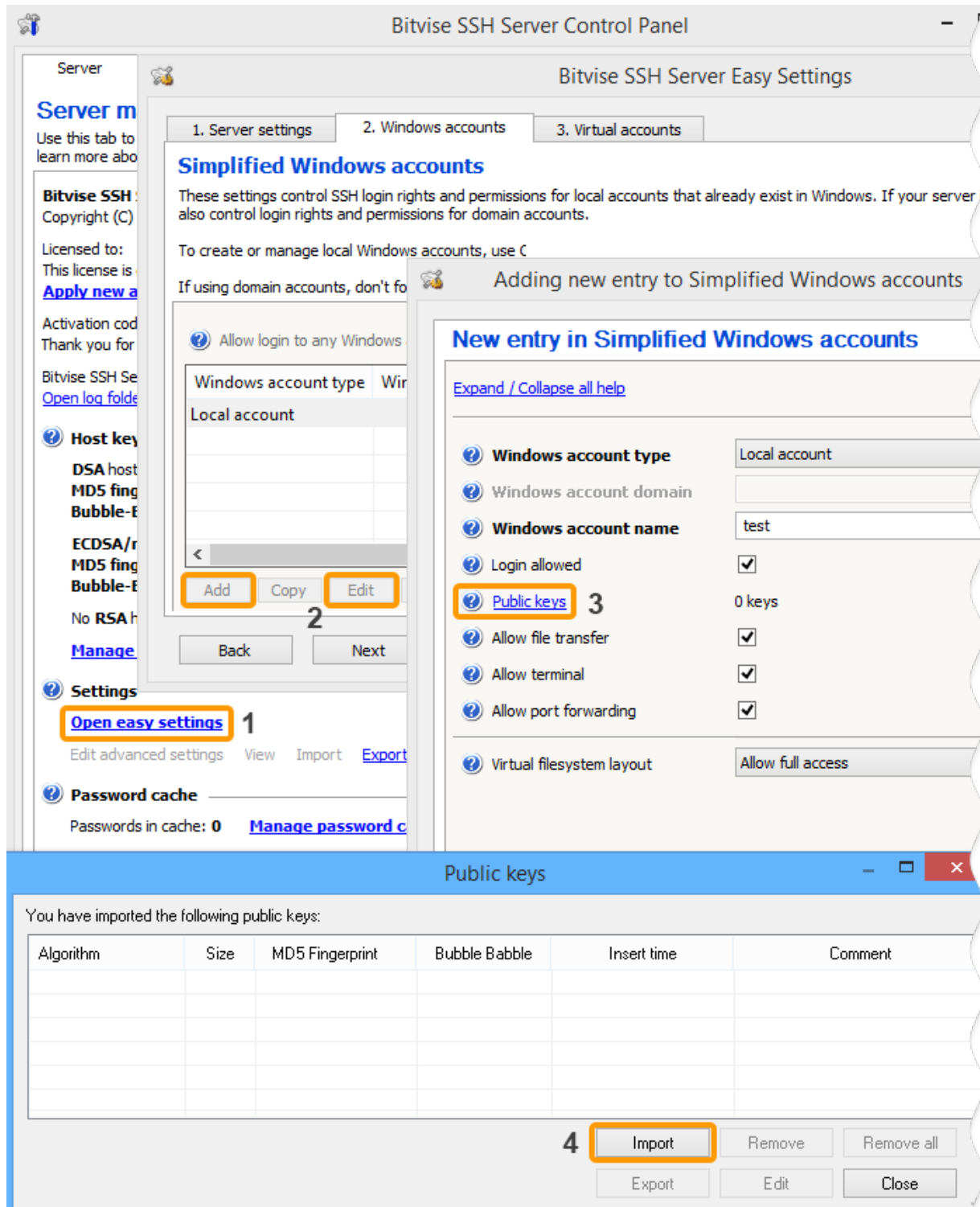
## Deploying public keys on Bitvise SSH Server

If you are new to Bitvise SSH Server (refer to the Bitvise documentation [↗](#) for specific configuration details), we highly recommend that you first make sure that you can establish a working SSH connection before you change any settings on the server. If you cannot connect to the SSH server using its default configuration, this is most likely due to a network or firewall problem that you will need to resolve before you are able to connect. In its default configuration, Bitvise SSH Server accepts connections on the often-used port number for SSH servers, 22. This is the only port that you need to open in your firewall in order to connect to the SSH server. If you use port forwarding to tunnel other applications through SSH, you should **not** open any additional ports for the tunnelled connections. All tunnelled connections are forwarded through the SSH session, established through port 22.

1. When connecting to Bitvise SSH Server with an SSH client for the first time, log in with the username and password of a Windows account that exists on the machine where the SSH server is running. To log into a Windows domain account, specify it in the `domain\account` format.

You can use any SSH client to log into Bitvise SSH Server, as long as it supports SSH protocol version 2.

2. Having ensured that the public key has been saved to a file, transfer it to the machine where Bitvise SSH Server is installed, or to the machine from which you manage the SSH Server remotely using Bitvise SSH Client.
3. Open the **SSH Server Control Panel**, and then, to import the public key into the SSH user's account settings, use either **Open easy settings**:



or **Edit advanced settings**:

**Bitvise SSH Server Control Panel**

**Server management**  
Use this tab to start and stop the server, and learn more about each configuration.

**Bitvise SSH Server 6.42**  
Copyright (C) 2000-2015 by Bitvise Inc.  
Licensed to: [Name]  
This license is good for business use.  
[Apply new activation code](#)  
Activation code valid for up to 30 days.  
Thank you for purchasing Bitvise SSH Server.  
Bitvise SSH Server service is running.  
[Open log folder viewer](#)

**Host keys**  
ECDSA/secp256k1 host key  
MD5 fingerprint: 2e:0a:11:11:11:11:11:11:11:11:11:11:11:11:11:11  
Bubble-Babble: xuzel  
SHA-256 fingerprint: [Fingerprint]  
ECDSA/nistp256 host key  
MD5 fingerprint: 8e:0a:11:11:11:11:11:11:11:11:11:11:11:11:11:11  
Bubble-Babble: xede  
SHA-256 fingerprint: [Fingerprint]  
No DSA and RSA host keys.  
[Manage host keys](#)

**Settings**  
[Open easy settings](#)  
[Edit advanced settings](#) 1

**Password cache**  
Passwords in cache: 0

**Configuration**  
Query

- Settings
  - Server
    - Bindings and ports
      - IPv4 (2)
      - IPv6 (0)
    - Windows Firewall
    - Usage status
    - Logging
    - Debugging
  - Algorithms
    - Key exchange
    - Encryption
    - Data integrity
    - Compression
  - Session
    - IP blocking -
    - IP blocking -
    - Windows domain
    - Proxy profiles (1)
  - Access control
    - Virtual accounts
    - Windows groups
    - Windows accounts
    - Virtual groups
    - Virtual accounts
    - Client version
    - Client address

**Windows accounts**  
A list of SSH entries for a Windows group will be added to the 'default' value also: [https://](#)

**Adding new entry to Windows accounts**

- New entry in Windows accounts
  - Limits and quotas
  - Client address
  - Authentication
  - Session setup
    - Windows file transfer
    - On-logon command
    - On-logoff command
    - On-upload
  - Terminal and environment
  - File transfer
    - Undefined mount point
  - Forwarding
    - Connect rule
    - Listening rule
    - Server connection
    - Server connection
    - Server connection

**Authentication**  
access.winAccounts.NewEntry  
Configure password and public account.  
[Expand / Collapse all help](#)

- ☐ Password authentication
- ☐ Allow password change
- ☐ Public key authentication
- ☐ Allow public key management
- ☒ **Public keys** 3

**Public keys**

You have imported the following public keys:

Algorithm	Size	MD5 Fingerprint	Bubble Babble	SHA-256 Fingerprint	Insert time	Comment

4 [Import](#) [Remove](#)  
[Export](#) [Edit](#)

**Note:**

For Windows accounts, Bitvise SSH Server also supports synchronisation with `~/.ssh/authorized_keys`, provided that this feature is enabled in **Advanced SSH Server settings**, under **Access control**. If this feature is enabled, Bitvise SSH Server will check for the existence of the `authorized_keys` file when the user logs out. If the file exists, Bitvise SSH Server will replace all the public keys configured for the user with the keys found in this file.

## Remote host access verification and WPS sign-on

1. Verify that login to the remote server can take place, for example:

```
jsmith@local-host$ ssh jsmith@remote-host
Last login: Wed Oct 21 17:22:33 2015 from 192.168.1.2
[Note: SSH did not ask for password.]

jsmith@remote-host$ [Note: You are on remote-host here]
```

**Note:**

If you are prompted for a password, then public key authentication has failed.

2. If public key authentication is successful, then, if you are using **WPS Link**, create the required host connection and remote host server through **WPS Workbench**. If you are using **WPS Communicate**, sign onto WPS using your private key, via the `SIGNON` statement, for which you need to specify either the **IDENTITYFILE** statement option or the **SSH\_IDENTITYFILE** system option, for example:

```
SIGNON <servername> SSH
USERNAME="<<username>"
IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";
```

Alternatively:

```
OPTIONS SSH_IDENTITYFILE="C:\Users\techwriter\.ssh\wpscommunicate.ppk";
SIGNON <servername> SSH
username="<<username>"
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr ";
```

**Note:**

You cannot use either **IDENTITYFILE** or **SSH\_IDENTITYFILE** if you are using *Passphrase authentication using ssh-agent* [↗](#) (page 69).

## Passphrase authentication using ssh-agent

**WPS Communicate** does not support the reading of private key files that have been saved in encrypted form with a **passphrase**. However, it is still possible to use private key-pairs of this form if you use a keychain agent. We will assume that you are using OpenSSH's **ssh-agent** for UNIX/Linux.

The **ssh-agent** program runs on the client system, acting as a temporary store of private keys in decrypted form. When the SSH client authenticates with a remote host, it can fetch the private key from the agent without needing to prompt the user.

On startup, **ssh-agent** does not hold any keys. These are loaded from disk using the **ssh-add** command, at which point the user enters each key's passphrase to decrypt it. The agent can store multiple keys simultaneously, from which the system will automatically choose the correct key for the remote server.

The following procedure assumes that you have already generated a key-pair with a **passphrase**, and deployed the public key on the remote SSH server (for UNIX/Linux), or on Bitvise SSH Server (for Windows).

Proceed as follows:

1. Start **ssh-agent** by typing the following into your local terminal session:

```
eval $(ssh-agent)
```

This will start the agent program and place it into the background.

---

**Note:**

The `eval` command tells the shell to run the output of **ssh-agent** as shell commands. Thereafter, processes run by this shell inherit its environment variables and have access to the agent.

---

2. Now, you need to add your private key to the agent, so that it can manage your key:

```
ssh-add
```

---

**Note:**

When run without arguments, **ssh-add** automatically adds the files `~/.ssh/id_rsa`, `~/.ssh/id_dsa`, `~/.ssh/id_ecdsa`, `~/.ssh/id_ed25519` and `~/.ssh/identity`.

---

3. You are prompted to enter your **passphrase**:

```
Enter passphrase for /home/demo/.ssh/id_rsa:
Identity added: /home/demo/.ssh/id_rsa (/home/demo/.ssh/id_rsa)
```

**Note:**

If you wish to be able to connect without a password to one server from within another server, you will need to forward your SSH key information. This will allow you to authenticate to another server through the server to which you are connected, using the credentials on your local host. To start this, **ssh-agent** must be running, and your private key must have been added to the agent (see above). You then need to connect to your first server using the `-A` option. This forwards your credentials to the server for this session:

```
ssh -A username@remote_host
```

From here, you can SSH into any other host that your SSH key is authorised to access. You will connect as if your private SSH key was located on this server.

**Note:**

WPS automatically detects whether or not **ssh-agent** is running.

## Kerberos single sign-on

You can set up Kerberos single sign-on so that you can sign on from WPS to a remote server without directly providing a password or SSH identity file.

This typically requires modifications to the configuration of the SSH daemon to accommodate Kerberos, details of which are outside the scope of this document. You will need to discuss the required modifications with the system administrator of the remote host, as setting up Kerberos authentication is a job for an experienced system administrator.

Kerberos configuration is also required on the client machine.

Before attempting to perform a Kerberos sign-on using WPS, you must be able to perform a Kerberos sign-on to a remote host using a conventional SSH client.

Once you can perform a Kerberos sign-on to a remote host using an external SSH client, you can perform the same sign-on from within WPS. You do not need to specify any authentication information with the `SIGNON` statement, but you should ensure that neither the **IDENTITYFILE** statement option nor the **SSH\_IDENTITYFILE** system option is used.

Example sign-on code might resemble:

```
SIGNON <servername> SSH  
LAUNCHCMD="/home/installs/wps-3.2/bin/wps -dmr";
```

## Linux clients

The `kinit` command will perform a Kerberos sign-on and prompt you for your Kerberos password. This action acquires a ticket which is cached for a period during which the ticket can be securely passed to other hosts as proof of identity, therefore allowing for authentication to those hosts without the need to provide further authentication information:

```
kinit  
ssh -o PreferredAuthentications=gssapi-with-mic <hostname>
```

Specifying the `PreferredAuthentications=gssapi-with-mic` option ensures that SSH only attempts GSSAPI authentication and does not, for example, attempt to use public key authentication which might otherwise proceed without prompting you for a password.

## Windows clients

It is possible to configure Kerberos single sign-on from WPS on Windows, but there are some restrictions:

- The first is that you **cannot** be a local administrator on your machine. If you are, Windows will not issue the Kerberos Ticket-Granting Ticket necessary for WPS to perform the Kerberos authentication.
- The second is that you need to set a registry key to allow Windows to give out the Ticket-Granting Ticket, even if you are not in the local administrators group. The value `allowtgtsessionkey` under the following key needs to be set to 1:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters
```

---

**Note:**

You may need to add this value to the registry if it is not already present.

---

---

**Note:**

The above restrictions do not apply to PuTTY, so, if you can log on using PuTTY but not WPS, then one of the above restrictions is likely to be the cause.

---

# Environment variables

Environment variables may need to be configured to provide access to third-party software being used with a particular WPS installation.

Using WPS with third-party applications such as database servers, requires that the environment information is available at WPS start up, such as:

- `LD_LIBRARY_PATH` or `LIBPATH` on Linux or Unix systems pointing to client libraries
- `ODBCSYSINI` pointing to the unixODBC client libraries
- `PATH`, for example pointing to the client libraries on Windows, or application directories on Linux or Unix systems.

The WPS recommended method for setting these environment variables is described in the `install-EN`.

Once the required environment variables have been set up by your system administrator, you can test whether a communicate session is available using the following SAS language program:

```
options ssh_hostvalidation=none;
%let host=host port;
SIGNON host ssh user="user" password="password"
  launchcmd="installation_path/bin/wps -dmr";

RSUBMIT;
  PROC OPTIONS;
  RUN;
ENDRSUBMIT;

SIGNOFF;
```

Where the target server hosting WPS is identified using the `HOST` macro variable, and your user credentials for that server are arguments to the `SIGNON` command. The `password` option for the `SIGNON` command is only required if you are using SSH password authentication; if you are using key-based authentication only a user identifier is required.

This example is for a communicate session connecting a Linux server; if you are using a Windows server, the `launchcmd` path will require altering. When run, this program connects to the identified server, invokes WPS on that server, displays the options set for the remote WPS server and then disconnects from the server.



# Legal Notices

Copyright © 2002–2019 World Programming Limited.

All rights reserved. This information is confidential and subject to copyright. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system.

## Trademarks

WPS and World Programming are registered trademarks or trademarks of World Programming Limited in the European Union and other countries. (r) or ® indicates a Community trademark.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

All other trademarks are the property of their respective owner.

## General Notices

World Programming Limited is not associated in any way with the SAS Institute.

WPS is not the SAS System.

The phrases "SAS", "SAS language", and "language of SAS" used in this document are used to refer to the computer programming language often referred to in any of these ways.

The phrases "program", "SAS program", and "SAS language program" used in this document are used to refer to programs written in the SAS language. These may also be referred to as "scripts", "SAS scripts", or "SAS language scripts".

The phrases "IML", "IML language", "IML syntax", "Interactive Matrix Language", and "language of IML" used in this document are used to refer to the computer programming language often referred to in any of these ways.

WPS includes software developed by third parties. More information can be found in the THANKS or acknowledgments.txt file included in the WPS installation.