## Contents

1 Overview .......................................................................................................................... 1

1.1 Introduction ................................................................................................................. 1

1.2 Technology Overview ................................................................................................. 2

1.2.1 Transport Module ............................................................................................... 3

1.2.2 User Authentication Module ............................................................................. 3

1.2.3 Connection Module ............................................................................................ 4

1.2.4 SSH File Transfer Protocol Module .................................................................. 4

1.2.5 RFC Conformance ............................................................................................. 4

1.3 Additional Documentation ....................................................................................... 5

Wind River Documentation ......................................................................................... 5

RFCs ......................................................................................................................... 6

2 Configuring and Building Wind River SSH ............................................................ 7

2.1 Introduction .............................................................................................................. 8

2.2 Configuring and Building Wind River SSH .......................................................... 8

2.3 Configuring VxWorks with Wind River SSH ...................................................... 8

2.3.1 Components and Parameters .......................................................................... 8
2.4 Building VxWorks with Wind River SSH ......................................................... 12
2.5 Building and Testing a Sample Project .............................................................. 13
  2.5.1 Recommended SSH Clients .................................................................... 14

3 Developing Wind River SSH Applications ........................................ 15
  3.1 Introduction ............................................................................................................. 15
  3.2 Developing Wind River SSH Kernel Applications ................................. 16
  3.3 Developing Wind River SSH RTP Applications .............................................. 17
  3.4 Initialization and Configuration Routines ....................................................... 17
  3.5 Authentication ........................................................................................................ 17
    3.5.1 Server Authentication .............................................................................. 18
    3.5.2 User Authentication ................................................................................. 19
    Public Key Authentication ........................................................................... 19
    Password Authentication .......................................................................... 19
  3.6 Run-time Configuration ..................................................................................... 20
  3.7 Integration with Shells ....................................................................................... 22
    3.7.1 Custom Shells ........................................................................................... 22
    3.7.2 Shell Integration Code Examples ........................................................... 22
    Shell Integration Using the C API Model .................................................... 23
    Shell Integration Using the Socket Model ................................................. 24
    3.7.3 Using Multiple Shells and Ports ............................................................. 25
  3.8 Starting and Stopping Wind River SSH ............................................................ 25
  3.9 SFTP Client Routines ......................................................................................... 25
  3.10 File Systems and SFTP ..................................................................................... 26
    3.10.1 Access Control ........................................................................................ 27
4 Command Line Usage ................................................................. 29
  4.1 Introduction ............................................................................. 29
  4.2 SSH Utilities ............................................................................ 30
    ipssh_list ............................................................................... 30
    ipssh_stop ........................................................................... 30
    keyfp .................................................................................. 31
  4.3 SFTP Client .............................................................................. 32
    sftp ..................................................................................... 32
  4.3.1 SFTP Client Sub-commands .............................................. 33
    cd ....................................................................................... 33
    get ..................................................................................... 33
    put ..................................................................................... 34
    pwd ..................................................................................... 35
    ls ....................................................................................... 35
    rm ....................................................................................... 35
    lcd ...................................................................................... 36
    lpwd .................................................................................... 36
    Ils ....................................................................................... 37
    lrm ...................................................................................... 37
    lmkdir ............................................................................... 37
    lrmdir ............................................................................... 38
    quit/exit ............................................................................. 38

A Libraries ..................................................................................... 39

B Routines ..................................................................................... 43

Index ............................................................................................. 73
1.1 Introduction

This guide describes Wind River SSH, which is an implementation of the Secure Shell protocol (SSH) that provides secure remote login, file transfer, and port forwarding over an insecure network.

Wind River SSH functions as both an SSH server and an SSH File Transfer Protocol (SFTP) server. It also includes an SFTP client.

Wind River SSH supports password and public key authentication for IPv4 and IPv6 connections as described in the 1.5 and 2.0 IETF draft versions of the Secure Shell (SSH) Protocol. Version 2.0 is not a superset of version 1.5, and each use their own support for various security features. For example, version 1.5 of the SSH Connection Protocol supports terminal connections and port forwarding. Version 2.0 provides this support, as well as SFTP support.
1.2 Technology Overview

The SSH protocol consists of the following components:

SSH Transport Protocol
This protocol provides a secure channel used by other SSH sub-protocols. The SSH Transport Protocol handles secure key exchange, server authentication, encryption, replay, and integrity protection.

SSH Authentication Protocol
This protocol is a general purpose protocol for user authentication. It requires a secure channel provided by the SSH Transport Protocol.

SSH Connection Protocol
This protocol provides interactive login sessions and forwards TCP connections. It can run the SSH File Transfer Protocol as a subsystem.

The following protocol is not a formal part of the SSH protocol suite, but it is designed to work in a secure environment such as that provided by SSH.

SSH File Transfer Protocol (SFTP)
This protocol provides the same services as the FTP protocol to provide secure file system access, with one main difference. In most cases, SFTP runs in a secure channel as a subsystem of the SSH Connection Protocol. Standard FTP runs over clear-text TCP connections.

Wind River SSH uses the cryptography services of the Wind River Cryptography Libraries.
1.2.1 **Transport Module**

Wind River SSH supports the SSH version 2.0 features listed in Table 1-1:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Supported Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Exchange</td>
<td>diffie-hellman-group1-sha1</td>
</tr>
<tr>
<td>Server Host Key</td>
<td>ssh-dsa</td>
</tr>
<tr>
<td>Encryption</td>
<td>3des-cbc, des-cbc, aes128-cbc, aes192-cbc, aes256-cbc, blowfish-cbc, cast128-cbc, arcfour, none</td>
</tr>
<tr>
<td>MAC</td>
<td>hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, none</td>
</tr>
</tbody>
</table>

Wind River SSH supports the SSH version 1.5 features listed in Table 1-2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Supported Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Host Key</td>
<td>ssh-rsa</td>
</tr>
<tr>
<td>Encryption</td>
<td>3des-cbc, des-cbc, blowfish-cbc, arcfour, none</td>
</tr>
</tbody>
</table>

See 3.5.1 *Server Authentication*, p.18 for more information about keys.

1.2.2 **User Authentication Module**

The SSH Authentication Protocol supports password authentication and public key authentication. Separate modules handle the validation of user ID, passwords, and public keys. These modules can be replaced. For more information, see the authentication APIs in the API reference entries. Both RSA and DSA keys are supported.

See 3.5.2 *User Authentication*, p.19 for more information about user authentication.
1.2.3 Connection Module

The connection module of Wind River SSH supports terminal connections as implemented by the protocol. Wind River SSH is pre-integrated with the VxWorks shell.

For more information about shells, integrating a proprietary shell with Wind River SSH, or using multiple shells, see 3.7 Integration with Shells, p.22.

1.2.4 SSH File Transfer Protocol Module

Wind River SSH supports both SSH File Transfer Protocol (SFTP) client and server. SFTP provides basic file management over the secure SSH connection. With this module, SSH supports:

- file transfers in both directions
- directory listings
- creation, removal, and management of directories and files

If the underlying file system does not support access control, you can implement a callback to handle access control on file operations. For more information, see 3.10 File Systems and SFTP, p.26.

1.2.5 RFC Conformance

Wind River SSH supports only the server end of terminal connections. Both SFTP client and server are supported. Wind River SSH conforms to the following RFCs, with conformance noted per RFC:

- RFC 4252: The Secure Shell (SSH) Authentication Protocol
  The following section of the protocol is not supported:
  - 9. Host-Based Authentication
- RFC 4253: The Secure Shell (SSH) Transport Protocol
  The following sections of the protocol are not supported:
  - 6.2. Compression
  - 9. Key Re-Exchange
1 Overview

1.3 Additional Documentation

- RFC 4254: The Secure Shell (SSH) Connection Protocol
  The following sections of the protocol are not supported:
  - 6.3. X11 Forwarding
  - 6.4. Environment Variable Passing
  - 6.5. Starting a Shell or a Command, exec message
  - 6.8. Local Flow Control
  - 6.9. Signals
  - 6.10. Returning Exit Status

- The Secure Shell (SSH) File Transfer Protocol, IETF Draft (draft-ietf-secsh-filexfer)
  Only SFTP version 3 is supported.

1.3 Additional Documentation

The following sections describe additional documentation about the technologies described in this book.

Wind River Documentation

The following Wind River documents present information associated with Wind River SSH:

- Wind River VxWorks Platforms Getting Started—describes how to build components of the Wind River VxWorks Platforms product.

- Wind River VxWorks Platforms Release Notes—describes reported and resolved software defects and new features for the Wind River VxWorks Platforms product.

- Wind River Cryptography Libraries Programmer's Guide—describes the cryptography routines used by Wind River SSH.
RFCs

The following IETF RFCs and draft document are relevant to Wind River SSH, and can be found at the IETF Web site http://www.ietf.org.

- RFC 4250: The Secure Shell (SSH) Protocol Assigned Numbers

- RFC 4251: The Secure Shell (SSH) Protocol Architecture

- RFC 4252: The Secure Shell (SSH) Authentication Protocol

- RFC 4253: The Secure Shell (SSH) Transport Protocol

- RFC 4254: The Secure Shell (SSH) Connection Protocol

- The Secure Shell (SSH) File Transfer Protocol, IETF Draft (draft-ietf-secsh-filexfer)
2

Configuring and Building
Wind River SSH

2.1 Introduction 8
2.2 Configuring and Building Wind River SSH 8
2.3 Configuring VxWorks with Wind River SSH 8
2.4 Building VxWorks with Wind River SSH 12
2.5 Building and Testing a Sample Project 13

This chapter describes VxWorks features; it does not go into detail about the mechanisms by which VxWorks-based systems and applications are configured and built. The tools and procedures used for configuration and build are described in the Wind River Workbench User’s Guide and the VxWorks Command-Line Tools User’s Guide.

NOTE: In this guide, as well as in the VxWorks API references, VxWorks components and their configuration parameters are identified by the names used in component description files. The names take the form, for example, of INCLUDE_FOO and NUM_FOO_FILES (for components and parameters, respectively).

You can use these names directly to configure VxWorks using the command-line configuration facilities.

Wind River Workbench displays descriptions of components and parameters, as well as their names, in the Components tab. You can use the Find dialog to locate a component or parameter using its name or description. To access the Find dialog from the Components tab, type CTRL+F, or right-click and select Find.
2.1 Introduction

This chapter describes how to build a VxWorks bootable image that includes Wind River SSH.

2.2 Configuring and Building Wind River SSH

Before Wind River SSH can be used in a VxWorks kernel application, it must be built as part of the top-level build for your Wind River Platform product. For information about this build, see the Wind River Platforms Getting Started.

2.3 Configuring VxWorks with Wind River SSH

2.3.1 Components and Parameters

Required Components

To include Wind River SSH in a kernel application, define the component INCLUDE_SSH. See Table 2-1 for a list of the configuration parameters for this component.

NOTE: Many of the configuration options can also be changed at run time. See 3.6 Run-time Configuration, p.20 for details.
### Table 2-1  Wind River SSH Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH_AUTH_MAX_FAIL</td>
<td>Controls the maximum number of authentication attempts before a client is disconnected. Enter a numeric value for this parameter. See 3.5.2 User Authentication, p.19 for details about authentication.</td>
<td>3</td>
</tr>
<tr>
<td>SSH_AUTH_PW_ALLOWED</td>
<td>Controls if password authentication is allowed. Enter 1 to use password authentication or 0 to disable this feature. See 3.5.2 User Authentication, p.19 for details about authentication.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_AUTH_PW_REQUIRED</td>
<td>Controls if password authentication is always required before a client is considered logged in. Enter 1 to use this requirement, or 0 to disable this feature. See 3.5.2 User Authentication, p.19 for details about authentication.</td>
<td>0</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_ALLOWED</td>
<td>Controls if public key authentication is allowed. Enter 1 to allow public key authentication, or 0 to disallow this functionality. See 3.5.2 User Authentication, p.19 for details about authentication.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_FIRST</td>
<td>Controls if public key authentication is performed first when both password and public key authentication are used. Enter 1 to set public key authentication as the first authentication method used before others, or 0 so the client can choose which order to authenticate. See 3.5.2 User Authentication, p.19 for details about authentication.</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_REQUIRED</td>
<td>Controls if public key authentication must be performed before a client is considered logged in. Enter 1 to require public key authentication, or 0 to not set this requirement. See 3.5.2 User Authentication, p. 19 for details about authentication.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_DSA_KEY</td>
<td>Sets the name of the DSA key file stored at IPCOM_FILE_ROOT. Enter the name of the DSA key file. See 3.5.1 Server Authentication, p. 18 for details about server keys.</td>
<td>dsa_key.pem</td>
</tr>
<tr>
<td>SSH_USE_PORT_FWD</td>
<td>Enables SSH port forwarding. Enter 1 to enable, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_SFTP</td>
<td>Enables SSH SFTP service. Enter 1 to enable, or 0 to disable this feature. When disabled, the SSH server will not accept client requests to open SFTP connections.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_TERMINAL</td>
<td>Enables SSH terminal service. Enter 1 to enable, or 0 to disable this feature. When disabled, the SSH server will not accept client requests to open terminal sessions.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_IDLE_TIMEOUT</td>
<td>Controls the number of seconds before an idle client is disconnected. Enter a numeric value for this parameter. Set to 0 for no timeout.</td>
<td>0</td>
</tr>
<tr>
<td>SSH_MAX_CLIENTS</td>
<td>Sets the maximum number of simultaneous clients. Enter a numeric value for this parameter.</td>
<td>20</td>
</tr>
<tr>
<td>SSH_MAX_SHELLS</td>
<td>Sets the maximum number of shells started per connection. Note: the VxWorks shell only supports one shell per SSH client.</td>
<td>1</td>
</tr>
</tbody>
</table>
### Wind River SSH Parameters (cont'd)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH_PORT</td>
<td>Sets the TCP port name or number that the SSH daemon listens to for incoming connections. At run time, the server can be configured to listen to additional ports. See 3.6 Run-time Configuration, p.20 for details.</td>
<td>22</td>
</tr>
<tr>
<td>SSH_RSA_KEY</td>
<td>Sets the name of the RSA key file stored at IPCOM_FILE_ROOT. See 3.5.1 Server Authentication, p.18 for details about server keys.</td>
<td>rsa_key.pem</td>
</tr>
<tr>
<td>SSH_USE_3DES</td>
<td>Enables the use of the 3DES algorithm. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_AES</td>
<td>Enables the use of the AES algorithm. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_BLOWFISH</td>
<td>Enables the use of the Blowfish algorithm. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_CAST</td>
<td>Enables the use of the CAST algorithm. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_DES</td>
<td>Enables the use of the DES algorithm. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
<tr>
<td>SSH_USE_NO_MAC</td>
<td>Sets a feature to not use any MAC algorithm authentication. Enter 1 to set no authentication, or 0 to not use this feature. This parameter is used to configure the SSH protocol for cleartext encryption that may be necessary for debugging purposes.</td>
<td>0</td>
</tr>
<tr>
<td>SSH_USE_MD5</td>
<td>Enables the use of the MD5 MAC algorithm for authentication. Enter 1 to use this algorithm, or 0 to disable this feature.</td>
<td>1</td>
</tr>
</tbody>
</table>
For information about building VxWorks with Wind River SSH, including build options, image types, and so on, see the Workbench User’s Guide and the VxWorks Command-line Tools User’s Guide.
2.5 Building and Testing a Sample Project

To check that the proper components are included in a VxWorks image, perform the following steps to build and test a sample project. These instructions do not provide complete details about VxWorks or running Wind River Workbench. They are intended as a demonstration of basic operation of Wind River SSH. Where necessary, refer to the appropriate guides for more details.

1. Build the platform source as described in the Getting Started guide for your platform. Be sure to include the necessary components for Wind River SSH.

2. Start Wind River Workbench. Create a new VxWorks image project. Set up the project based on the simpc board support package. See the Wind River Workbench User’s Guide for details.

3. In the kernel configuration, include the INCLUDE_SSH component. Leave all parameters at their default values.

4. Add a user ID and password. In the kernel configuration, include the INCLUDE_IPCOM_AUTH_1 component. Set the AUTH_USER_NAME1 parameter to “ssh”. Set the AUTH_USER_PASSWD1 parameter to “secret”.

5. Build the project. This creates a VxWorks image.

6. Ensure that the network daemon vxsimnetd is running. See the Wind River VxWorks Simulator User’s Guide for details.

7. In Workbench, create a new target connection of type VxWorks Simulator Connection. Set the boot file name to a custom simulator. Browse to the VxWorks image for the project.

8. Under Advanced Boot Parameters, set the boot device to simnet. Set the IP address to 192.168.200.2. See the Wind River Workbench User’s Guide for details.

9. Connect to the simulator. The VxWorks Kernel Shell appears.

At this point, the simulator is running the SSH server and is ready to accept connections. Using an SSH client, connect to the SSH server using the following parameters:

- IP address: 192.168.200.2
- Port: 22
- Protocol: SSH
- User ID: ssh
- Password: secret
See 2.5.1 Recommended SSH Clients, p.14 for a list of suitable SSH client software packages.

2.5.1 Recommended SSH Clients

OpenSSH is the recommended SSH client for UNIX dialects. For more information, see http://www.openssh.org.

The following SSH clients are recommended for Windows clients:

- The SSH and SFTP clients of cygwin bash. For more information, see http://www.cygwin.com
- PuTTY. For more information, see http://www.chiark.greenend.org.uk/~sgtatham/putty/
- Mindterm 2.x. For more information, see http://www.mindbright.se
3.1 Introduction  
Wind River SSH applications must be created as kernel applications that execute in kernel-memory space in kernel mode. They cannot be created as a real-time process (RTP).
The Wind River SSH server runs as a daemon process that starts at boot time. Generally, modifying the way the daemon operates requires changing configuration options and modifying initialization and callback routines.

Wind River SSH also includes an SFTP client with its own API. The SFTP client can also be run in the kernel shell using the command interpreter.

### 3.2 Developing Wind River SSH Kernel Applications

VxWorks applications that execute in the kernel are created as relocatable object modules. When you build a kernel-based application module, user code is linked to the required VxWorks libraries, and an ELF binary is produced. Kernel-based application modules use VxWorks facilities by including header files that define VxWorks interfaces and data structures.

Kernel-based application modules can be either of the following:

- Downloaded and dynamically linked to VxWorks by the object module loader.
- Statically linked to VxWorks, making them part of the system image.

For information about creating kernel applications, see the following manuals:

- VxWorks Kernel Programmer’s Guide
- Wind River Workbench User’s Guide

Wind River SSH includes a server for SSH and SFTP, and an SFTP client. Modifications to server functionality can be done using a combination of the following techniques:

- Change kernel configuration parameters, as described in 2. Configuring and Building Wind River SSH.
- Add initialization and configuration code to `ipssh_configure()`, as described in 3.4 Initialization and Configuration Routines, p.17.
- Change callback routines that implement specific functions. For example, user authentication is handled by callback routines for public key authentication and password authentication as described in 3.5 Authentication, p.17.
- Dynamically change configuration at runtime, as described in 3.6 Run-time Configuration, p.20.
The SFTP client API can be used to connect to an external SFTP server. See 3.9 SFTP Client Routines, p.25 for details.

3.3 Developing Wind River SSH RTP Applications

Wind River SSH cannot run as a real-time process. For information about kernel mode applications, see 3.2 Developing Wind River SSH Kernel Applications, p.16.

3.4 Initialization and Configuration Routines

The Wind River SSH daemon starts at boot time. ipssh_config.c, in the directory installDir/components/ip_net2-6.x/osconfig/vxworks/src/ipnet, contains the routine ipssh_configure(), which performs initialization before the daemon starts. Modify this routine to change initialization options.

The file ipssh_config.c also contains callback routines that can be used to change run-time operation. Modify the routines as required. The callback routines are described in the API reference entries. The names of the callback routines all end with _cb, like ipssh_validate_pubkey_cb().

3.5 Authentication

The SSH protocol uses different types of authentication at different stages of the client-server negotiation.

As part of the transport protocol, the client authenticates the server by means of a public key. Most clients will store a list of trusted servers and their keys, and provide a means for adding new servers to that list.
Once the transport has been set up, the server authenticates the client to determine
whether it will grant access, and what privileges are available. This is normally
done through a user name and password combination or through a public key
presented by the client. For public key authentication, the server must have a way
of storing trusted client user names and their associated keys.

### 3.5.1 Server Authentication

The SSH protocol requires the server to have a server key pair for authentication
during transport negotiation. Wind River SSH uses either a Digital Signature
Algorithm (DSA) or a public key algorithm (RSA) key pair.

**NOTE:** The default implementation of Wind River SSH is suitable for testing
purposes only, because it hard-codes the private key. `IPSSH_USE_DEFAULT_KEYS`,
defined in `installDir/components/ip_net2-6.x/ipssh/config/ipssh_config.h`,
determines whether the default keys are compiled into the image. To change the
default behavior, and use your own generated keys, remove the definition for
`IPSSH_USE_DEFAULT_KEYS`.

The default key values are defined in
`installDir/components/ip_net2-6.x/osconfig/vxworks/src/ipnet/ipssh_config.c`

When default keys are not being used, appropriate keys must be generated and
stored on the server. The kernel configuration parameters `SSH_DSA_KEY` and
`SSH_RSA_KEY` define file names that contain the private keys, in privacy enhanced
mail (PEM) format. The files must be located in the `IPCOM_FILE_ROOT` directory
(`/ram/`).

Use any suitable method to generate and store the private keys. Depending on the
needs of the application, this could mean generating the keys externally and
loading them onto the target, or it could mean generating the keys directly on the
target using cryptography routines available as part of the Wind River
Cryptography Libraries.
3.5.2 User Authentication

The SSH protocol allows different methods to be used for user authentication. The most common methods are with a public key or using a password.

To set the types of user authentication that will be used, and in what order, set the following kernel configuration parameters:

- SSH_AUTH_PW_ALLOWED
- SSH_AUTH_PW_REQUIRED
- SSH_AUTH_PUB_KEY_ALLOWED
- SSH_AUTH_PUB_KEY_REQUIRED
- SSH_AUTH_PUB_KEY_FIRST

Public Key Authentication

For public key authentication, the callback routine ipssh_validate_pubkey_cb() is used to validate a public key sent by an SSH client. The default behavior checks for the existence of a file named user.pk2 in the IPCOM_FILE_ROOT directory (/ram/), where user is the user name to validate. This public key is used to validate a signed data stream created using the client’s private key.

Note that the server must have copies of the public keys for clients before public key authentication will be successful. If the server does not have a matching key it notifies the client, which can then try password authentication.

To change the method of authentication, modify ipssh_validate_pubkey_cb(). For example, you could change how public keys are stored and retrieved.

Password Authentication

The callback routine ipssh_validate_userid_pw_cb() is used to validate a password sent by an SSH client. The default behavior checks the user name/password combination against the values of the AUTH_USER_NAME and AUTH_USER_PASSWD kernel configuration parameters. For example, to allow access for user “sshuser” with password “secret”, set AUTH_USER_NAME to “sshuser” and AUTH_USER_PASSWD to “secret” in the kernel configuration.

To change how user names and passwords are managed, modify ipssh_validate_userid_pw_cb(). For example, the server could store hash values for passwords instead of plain text values.
3.6 Run-time Configuration

The configuration parameters for the INCLUDE_SSH component have corresponding system variables (sysvars) that can be used to modify the run-time configuration of Wind River SSH. For example, password authentication can be enabled or disabled dynamically by changing the value of the ipssh.auth.pw.allowed sysvar.

**NOTE:** Changes to sysvars will only take effect after the ipssh daemon has been reconfigured, using either the ipd shell command or the ipd API.


Table 3-1 lists the INCLUDE_SSH configuration parameters and their corresponding sysvars.

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Sysvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH_AUTH_MAX_FAIL</td>
<td>ipssh.auth.max_fail</td>
</tr>
<tr>
<td>SSH_AUTH_PW_ALLOWED</td>
<td>ipssh.auth.pw.allowed</td>
</tr>
<tr>
<td>SSH_AUTH_PW_REQUIRED</td>
<td>ipssh.auth.pw.required</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_ALLOWED</td>
<td>ipssh.auth.pub_key.allowed</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_FIRST</td>
<td>ipssh.auth.pub_key_first</td>
</tr>
<tr>
<td>SSH_AUTH_PUB_KEY_REQUIRED</td>
<td>ipssh.auth.pub_key.required</td>
</tr>
<tr>
<td>SSH_DSA_KEY</td>
<td>ipssh.file.dsa_key</td>
</tr>
<tr>
<td>SSH_USE_PORT_FWD</td>
<td>ipssh.service.port_fwd</td>
</tr>
<tr>
<td>SSH_USE_SFTP</td>
<td>ipssh.service.sftp</td>
</tr>
<tr>
<td>SSH_USE_terminal</td>
<td>ipssh.service.terminal</td>
</tr>
<tr>
<td>SSH_IDLE_TIMEOUT</td>
<td>ipssh.idle_timeout</td>
</tr>
<tr>
<td>SSH_MAX_CLIENTS</td>
<td>ipssh.max_clients</td>
</tr>
</tbody>
</table>
### Table 3-1  Configuration Parameters and Sysvars (cont’d)

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Sysvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH_MAX_SHELLS</td>
<td>ipssh.max_shells</td>
</tr>
<tr>
<td>SSH_PORT</td>
<td>ipssh.port. If required, additional ports can be specified using the sysvars ipssh.port2, ipssh.port3, and so on. Note that ipssh.port1 is not a valid sysvar.</td>
</tr>
<tr>
<td>SSH_RSA_KEY</td>
<td>ipssh.file.rsa_key</td>
</tr>
<tr>
<td>SSH_USE_3DES</td>
<td>ipssh.enc.3des</td>
</tr>
<tr>
<td>SSH_USE_AES</td>
<td>ipssh.enc.aes</td>
</tr>
<tr>
<td>SSH_USE_BLOWFISH</td>
<td>ipssh.enc.blowfish</td>
</tr>
<tr>
<td>SSH_USE_CAST</td>
<td>ipssh.enc.cast</td>
</tr>
<tr>
<td>SSH_USE_DES</td>
<td>ipssh.enc.des</td>
</tr>
<tr>
<td>SSH_USE_NO_MAC</td>
<td>ipssh.mac.none</td>
</tr>
<tr>
<td>SSH_USE_MD5</td>
<td>ipssh.mac.md5</td>
</tr>
<tr>
<td>SSH_USE_MD5_96</td>
<td>ipssh.mac.md5-96</td>
</tr>
<tr>
<td>SSH_USE_NO_CIPHER</td>
<td>ipssh.enc.none</td>
</tr>
<tr>
<td>SSH_USE_ARCFOUR</td>
<td>ipssh.enc.arcfour</td>
</tr>
<tr>
<td>SSH_USE_SHA1</td>
<td>ipssh.mac.sha1</td>
</tr>
<tr>
<td>SSH_USE_SHA1_96</td>
<td>ipssh.mac.sha1-96</td>
</tr>
<tr>
<td>SSH_USE_V1</td>
<td>ipssh.v1</td>
</tr>
<tr>
<td>SSH_USE_V2</td>
<td>ipssh.v2</td>
</tr>
</tbody>
</table>
3.7 Integration with Shells

By default, Wind River SSH is completely integrated with the VxWorks shell. A user that has been successfully authenticated has a terminal connection to the VxWorks shell, in the same way as when logging in using telnet. All registered shell commands are available in an SSH terminal connection.

3.7.1 Custom Shells

To use a custom shell instead of the default, modify `ipssh_shell_start_cb()`. A typical shell workflow occurs as follows:

1. When a client logs in and requests a shell, SSH calls `ipssh_shell_start_cb()` to start a shell.
2. During the session SSH decrypts data from the client and sends it in clear text to the shell’s standard input.
3. The shell can also send data from its standard output to SSH, which encrypts it and sends it to the SSH client.
4. If the client disconnects, SSH notifies the shell. If the shell terminates, it can order the release of the SSH connection by calling SSH.

3.7.2 Shell Integration Code Examples

You can integrate a shell with Wind River SSH either using a C API or a socket connection between SSH and the shell.

The following code examples describe standard-input and standard-output communication in the shell process. Use these code examples to program a simple shell or one that integrates with a socket model. The examples modify callback routines in the following file:

`InstallDir/components/ip_net2-6.x/osconfig/vxworks/src/ipnet/ipssh_config.c`
Shell Integration Using the C API Model

Using this model, the data between Wind River SSH and the shell is conveyed in traditional C calls. Example 3-1 describes a shell that simply echoes any data received on standard input. If a ‘Q’ is received the connection is released.

For more details on these functions, see the API reference entries.

Example 3-1 Simple Shell Data Example

```c
void*
ipssh_shell_start_cb(void* ssh_ctx, const char* user, void* cookie)
{
    static shell_ctx = 0;
    shell_ctx++;
    return shell_ctx;
}

Ip_err
ipssh_shell_stdin_cb(void* ssh_ctx, void* shell_ctx,
    const char* data, int len)
{
    int ret;
    switch(data[0])
    {
    case 'Q':
        ipssh_shell_exit(ssh_ctx, shell_ctx);
        return IPCOM_SUCCESS;
    }
    ret = ipssh_send_shell_stdout_data(ssh_ctx,
        shell_ctx,
        data,
        len);
    if(ret > 0)
        return IPCOM_SUCCESS;
    else
        return IPCOM_ERR_FAILED;
}

Ip_err
ipssh_shell_stop_cb(void* ctx, void* shell_ctx)
{
    return IPCOM_SUCCESS;
}
```
Shell Integration Using the Socket Model

The shell socket API is implemented using the C API described in Example 3-1. When this model is used, SSH interacts with the shell using a TCP connection. Standard-input data sent from the client is forwarded to the shell over the TCP connection. In the same manner, data output from the shell on standard output must be sent to SSH over the TCP connection.

The SSH module closes the TCP connection if the SSH client disconnects. And equivalently, if the shell closes the TCP connection, SSH disconnects the SSH client.

This model is useful if you have an existing telnet server and want to use an SSH connection instead of a telnet connection. If you are using this model, however, a function that establishes a TCP connection to the telnet server must be implemented. Thereafter, you can connect to the system using SSH, and use the shell for the telnet connection.

The telnet server typically needs to be modified to listen on localhost only. For security reasons, it is important that this connection only establish over the loopback interface (localhost), and any authentication must be turned off in the telnet server as SSH handles the authentication.

**NOTE:** Wind River SSH configuration uses a socket model to integrate the VxWorks shell.

As described in Example 3-2, you must replace `ipssh_sock_shell_start_cb()` in `ipssh_config.c` to integrate a proprietary shell using this model.

```c
IP_PUBLIC Ip_err
ipssh_sock_shell_start_cb(Ip_fd *stdio_sock, Ip_fd client_fd)
{
    /* Start shell, establish a connection to the shell's stdio and set
       stdio_sock to the file descriptor of the connection */
    return IPCOM_SUCCESS;
}
```

**NOTE:** The socket descriptor returned in `ipssh_sock_shell_start_cb()` must be created by calling `ipcom_socket()` as opposed to calling `socket()`.

Using this model, you can easily convert an existing telnet server to an SSH server as no other shell management callbacks must be updated. For more information, see the API reference entries.
3.7.3 Using Multiple Shells and Ports

Wind River SSH supports multiple shells or the ability to listen to multiple ports. You can configure the ports that a system listens to using system variables. For more information, see 3.6 Run-time Configuration, p.20.

To start different shells based on the port the client uses to connect, modify the callback routine `ipssh_shell_start_cb()`. Call `ipcom_getpeername()` to get the port number.

If you are using a C API model, the client socket descriptor is obtained by calling `ipssh_get_clt_fd()`. For more information, see the API reference entry for `ipssh_get_clt_fd()`.

3.8 Starting and Stopping Wind River SSH

Wind River SSH is automatically started at boot.

You can start and stop the SSH server module, however, using the `ipd` framework of IPCOM. The framework includes a shell command, `ipd`, as well as a C API defined in `ipcom_ipd.h`.

3.9 SFTP Client Routines

Wind River SSH includes an SFTP client for connecting to an external SFTP server. The API for the client includes routines for standard file transfer operations such as connecting to a server and manipulating files and directories.

All the routines use an `lpssh_conn_st` struct to manage the connection to the server.

To create a new connection, perform the following steps:

1. Call `ipssh_sftp_clt_set_userid()` to set the user name for the connection.
2. Call `ipssh_sftp_clt_set_pw()` or `ipssh_sftp_clt_set_key()` to set either the password or public key.

3. Call `ipssh_sftp_clt_open()` to open the connection.

The other `ipssh_sftp_clt_*()` routines use the open connection. See the API reference entries for details.

### 3.10 File Systems and SFTP

You may wish to store or retrieve files from sources other than a traditional file system. For example, when you download updated software to a system, it is more efficient to download the new software version directly to memory without temporarily storing it in a local file system.

As another example, when large quantities of logging data are kept in memory, this must be uploaded to another system. In this case, it is convenient to retrieve the data directly from RAM without temporarily storing it as a file in the local file system.

Wind River SSH uses a file system API to support the use of other file sources and destinations. If your application needs to download or retrieve files from another source, other than a traditional file system, you can easily implement the API provided for this purpose.

Wind River SSH is distributed with one example implementation of the SFTP file API. This implementation uses a traditional file system as the source and destination of file operations. For more information, see `ipssh_sftp_file.c` in the directory `installDir/components/ip_net2-6.x/ipssh`.

Use `ipssh_sftp_file.c` as a reference when implementing this API for other file sources.

Wind River SSH calls `ipssh_get_sftp_func_cb()` for each new file request. This function returns a set of function pointers to the file system that handles file operations for the path identified in the request. `ipssh_get_sftp_func_cb()` is defined in `ipssh_config.c`. The default implementation returns pointers to routines from `ipssh_sftp_file.c`, but it can be modified to return different sets of function pointers for different directories.
3.10.1 **Access Control**

The callback routine `ipssh_sftp_check_access_cb()`, defined in `ipssh_config.c`, can be used to implement access control for SFTP operations. Modify the routine to restrict access to files and directories based on user name.
4

Command Line Usage

4.1 Introduction 29
4.2 SSH Utilities 30
4.3 SFTP Client 32

4.1 Introduction

Wind River SSH includes command-interpreter commands for the following tasks:

- Listing all logged in clients and files.
- Stopping a spawned SSH server process or all spawned processes.
- Generating a key fingerprint of an RSA or a DSA public key.

It also includes a command-interpreter command to run an SFTP client.

For basic information on the kernel shell, C interpreter, and command-interpreter commands, see the VxWorks Kernel Programmer’s Guide: Target Tools.
4.2 SSH Utilities

The SSH utilities are VxWorks shell commands to list SSH clients, stop SSH processes, and display fingerprints for RSA and DSA keys.

**ipssh_list**

**Name**

`ipssh_list`—list all logged in clients

**Synopsis**

`ipssh_list [-l] [-h]`

**Description**

This command lists all logged in clients.

The shell command options are as follows:

- `-l`
  
  print long listings, including information about the authentication method and encryption algorithm for each client.

- `-h`
  
  print usage.

**Example**

```
[vxWorks *]# ipssh_list -l
Connected clients:
  ID  User  Clt Addr  Service  Auth Meth  Encr Alg  MAC Alg  Clt SW
1635127272 ipnet ::FFFF:192.168.200.254 terminal password aes128-cbc hmac-md5 OpenSSH_3.9p1
```

**ipssh_stop**

**Name**

`ipssh_stop`—stop a particular spawned SSH server process or all spawned processes

**Synopsis**

`ipssh_stop [-a] [-c clt_id] [-h]`
4 Command Line Usage
4.2 SSH Utilities

Description

This command stops a particular spawned SSH server process or all spawned processes. Stopping a spawned SSH server process has the effect of logging out the corresponding client.

The shell command options are as follows:

- **-c clt_id**
  - stop the spawned SSH server process, with client id equal to clt_id. See the ipssh_list command for how to list client IDs.

- **-a**
  - stop all spawned SSH server processes.

- **-h**
  - print usage information.

Example

The following command stops all spawned processes, thus logging out all clients:

```
ipssh_stop -a
```

keyfp

Name

**keyfp**—generate a key fingerprint of an RSA or a DSA public key

Synopsis

```
keyfp -f file {-h|-b}
```

Description

This command generates a key fingerprint of an RSA or a DSA public key. The shell command options are as follows:

- **-f file**
  - The file holding the key to generate a fingerprint for.

- **-h**
  - Generate a hex fingerprint.

- **-b**
  - Generate a bubble-babble fingerprint.
4.3 SFTP Client

The `sftp` command, described below, creates a connection to an SFTP server, and starts a subsystem to process the other SFTP client commands.

sftp

Name

`sftp`—establish a connection to an SFTP server

Synopsis

```
sftp -u user [-i file] [-p port] [-h] host
```

Description

This command establishes a connection to an SFTP server. After the shell connection is established, an sftp prompt represents output, and the sftp shell command waits for input of shell sub-commands described in 4.3.1 SFTP Client Sub-commands, p.33. The command options are as follows:

- `u user` 
  user ID to use during authentication.

- `i file` 
  file holding a key pair to be used during authentication. If a key file is not specified the user is prompted for a password.

- `p port` 
  port to which the SFTP server listens.

- `h` 
  print usage.

- `host` 
  IP address hostname of the SFTP server.

Example

This example establishes an SFTP connection for `dude` with a `dsa_key.pem` key file on port 22 of a particular IP address.

```
> sftp -u dude -i dsa_key.pem -p 22 -h 10.31.0.0
```
4.3.1 SFTP Client Sub-commands

The following sub-commands are valid in an SFTP client session.

**cd**

**Name**

cd—change the working directory on the SFTP server

**Synopsis**

```
cd path
```

**Description**

This command changes the working directory on the SFTP server. A path can be either absolute or relative when changing directories. The shell command option `path` represents the new working directory.

**Example**

To change directory using an absolute path for a file source destination.

```bash
> cd /usr/local
```

To change directory using a relative path for a file source destination.

```bash
> cd bin
```

**get**

**Name**

get—transfer a file from the SFTP server to the local file system

**Synopsis**

```
get remote-path [local-path]
```

**Description**

This command transfers a file from the SFTP server to the local file system. The `get` shell command options are:
remote-path
the path to the file on the SFTP server.

local-path
the path to the file on the local file system. If no local path is given, remote-path is used.

Example
To transfer work.c from the SFTP server to the local file system:
> get work.c
or
> get work.c /home/tom/down_loaded_work.c

put

Name
put—transfer a file from the local file system to the SFTP server

Synopsis
put local-path [remote-path]

Description
This command transfers a file from the local file system to the SFTP server. The shell command options are as follows:
local-path
the path to the file on the local file system.
remote-path
the path to the file on the SFTP server. If no remote path is given, local-path is used.

Example
To transfer a file from the local file system to the SFTP server:
> put work.c
or
> put work.c /my_dir/uploaded_work.c
**pwd**

**Name**

`pwd`—print the current working directory on the SFTP server

**Synopsis**

```
pwd
```

**Description**

This command prints the current working directory on the SFTP server.

**Example**

To print the current working directory on the SFTP server:

```
> pwd
```

**ls**

**Name**

`ls`—list the files on SFTP server

**Synopsis**

```
ls
```

**Description**

This command lists the files on SFTP server.

**rm**

**Name**

`rm`—delete a file from the SFTP server

**Synopsis**

```
rm file
```
Description

This command deletes a file from the SFTP server. The shell command option `file` represents the file to remove.

Example

To delete a file from the SFTP server

```bash
> rm work.c
```

**lcd**

Name

`lcd`—change the local working directory

Synopsis

`lcd path`

Description

This command changes the local working directory. The shell command option `path` represents the new local working directory.

Example

To change the local working directory:

```bash
> lcd h/project/new
```

**lpwd**

Name

`lpwd`—print local working directory

Synopsis

`lpwd`

Description

This command prints the local working directory.
Ils

Name
Ils—print local directory listing

Synopsis
Ils

Description
This command prints the local directory listing.

lrm

Name
lrm—remove a file from local file system

Synopsis
lrm path

Description
This command removes a file from the local file system. The shell command option path represents the local file to remove.

Example
To remove a file from the local file system:
> lrm p/SSH/system/files

lmkdir

Name
lmkdir—create a new local directory

Synopsis
lmkdir dir
Description

This command creates a new local directory. The shell command option \textit{dir} represents the name of the directory to create.

Example

To create a new local directory called \textit{m}:

\begin{verbatim}
> lmkdir m
\end{verbatim}

\textbf{lrmdir}

Name

\textbf{lrmdir}—remove a local directory

Synopsis

\begin{verbatim}
lrmdir dir
\end{verbatim}

Description

This command removes the local directory. The shell command option \textit{dir} represents the name of the directory to remove.

Example

To remove the local directory:

\begin{verbatim}
> lrmdir dir m
\end{verbatim}

\textbf{quit/exit}

Name

\textbf{quit/exit}—disconnect from the SFTP server

Synopsis

\begin{verbatim}
quit
exit
\end{verbatim}

Description

This command disconnects from the SFTP server.
ipssh – Public API of Wind River SSH  40
ipssh_config – Configuration API of Wind River SSH  40
ipssh_sftp_clt – Public SFTP API of Wind River SSH  41
ipssh

NAME
ipssh – Public API of Wind River SSH

ROUTINES
ipssh_conn_new() – create a new SSH connection handle
ipssh_stop_spawn() – stop a spawned process
ipssh_stop_all spawns() – stop all spawned IPSSH tasks
ipssh_evp_pkey_cmp() – compare two public keys
ipssh_get_clt_info() – return information on logged-in client
ipssh_send_shell_stdout_data() – send stdout data to the SSH client
ipssh_shell_exit() – report that a shell has terminated
ipssh_key_fingerprint() – generate a fingerprint of a public key
ipssh_get_clt_fd() – return the socket descriptor of a client
ipssh_get_sftp_func_cb() – perform file operations on a specified file

DESCRIPTION
This library contains the API for Wind River SSH.

INCLUDE FILES
none

ipssh_config

NAME
ipssh_config – Configuration API of Wind River SSH

ROUTINES
ipssh_load_dsa_key() – return the DSA server key to be used by the daemon
ipssh_load_rsa_key() – return the RSA server key to be used by the daemon
ipssh_sftp_check_access_cb() – check user’s access to file system
ipssh_configure() – function to place any custom initialization code in.
ipssh_shell_start_cb() – start a shell at user login
ipssh_sock_shell_start_cb() – create shell with socket connection
ipssh_shell_welcome_msg_cb() – display welcome message at shell creation
ipssh_shell_stdin_cb() – decrypt standard-input data and send to shell
ipssh_shell_stop_cb() – terminate the shell
ipssh_validate_userid_pw_cb() – validate user’s login ID and password
ipssh_validate_pubkey_cb() – validate client’s public key in SSH v2
ipssh_validate_rsa_pubkey_v1_cb() – validate client public key in SSH v1

DESCRIPTION
This library contains the APIs used for configuration of Wind River SSH.

INCLUDE FILES
none
A Libraries

ipssh_sftp_clt

NAME

ipssh_sftp_clt – Public SFTP API of Wind River SSH

ROUTINES

- ipssh_sftp_clt_open() – open a connection to an SFTP server
- ipssh_sftp_clt_get() – get a file from the server
- ipssh_sftp_clt_put() – transfer a file to the server
- ipssh_sftp_clt_close() – close a connection
- ipssh_sftp_clt_set_userid() – set user ID for authentication
- ipssh_sftp_clt_set_pw() – set password for authentication
- ipssh_sftp_clt_set_key() – set key pair for authentication
- ipssh_sftp_clt_get_realpath() – convert a relative path to an absolute path
- ipssh_sftp_clt_get_cwd() – get current working directory on the server
- ipssh_sftp_clt_chdir() – change current working directory on the server
- ipssh_sftp_clt_mkdir() – create a directory on the server
- ipssh_sftp_clt_rmdir() – delete a directory on the server
- ipssh_sftp_clt_opendir() – open a directory stream on a server
- ipssh_sftp_clt readdir() – read the next directory entry on the server
- ipssh_sftp_clt_closedir() – close a directory stream on the server
- ipssh_sftp_clt_rename() – rename a file on the server
- ipssh_sftp_clt_unlink() – delete a file on the server

DESCRIPTION

This library contains the SFTP API for Wind River SSH.

INCLUDE FILES

none
Routines

ipssh_configure() – function to place any custom initialization code in. 45
ipssh_conn_new() – create a new SSH connection handle 45
ipssh_evp_pkey_cmp() – compare two public keys 45
ipssh_get_clt_fd() – return the socket descriptor of a client 46
ipssh_get_clt_info() – return information on logged-in client 47
ipssh_get_sftp_func_cb() – perform file operations on a specified file 47
ipssh_key_fingerprint() – generate a fingerprint of a public key 48
ipssh_load_dsa_key() – return the DSA server key to be used by the daemon 49
ipssh_load_rsa_key() – return the RSA server key to be used by the daemon 49
ipssh_send_shell_stdout_data() – send stdout data to the SSH client 50
ipssh_sftp_check_access_cb() – check user's access to file system 51
ipssh_sftp_clt_chdir() – change current working directory on the server 52
ipssh_sftp_clt_close() – close a connection 52
ipssh_sftp_clt_closedir() – close a directory stream on the server 53
ipssh_sftp_clt_get() – get a file from the server 54
ipssh_sftp_clt_get_cwd() – get current working directory on the server 54
ipssh_sftp_clt_get_realpath() – convert a relative path to an absolute path 55
ipssh_sftp_clt_mkdir() – create a directory on the server 56
ipssh_sftp_clt_open() – open a connection to an SFTP server 57
ipssh_sftp_clt_opendir() – open a directory stream on a server 58
ipssh_sftp_clt_put() – transfer a file to the server 58
ipssh_sftp_clt_readdir() – read the next directory entry on the server 59
ipssh_sftp_clt_rename() – rename a file on the server 60
ipssh_sftp_clt_rmdir() – delete a directory on the server 61
ipssh_sftp_clt_set_key() – set key pair for authentication 61
ipssh_sftp_clt_set_pw() – set password for authentication 62
ipssh_sftp_clt_set_userid() – set user ID for authentication 62
ipssh_sftp_clt_unlink() – delete a file on the server 63
ipssh_shell_exit() – report that a shell has terminated 64
ipssh_shell_start_cb() – start a shell at user login 64
ipssh_shell_stdin_cb() – decrypt standard-input data and send to shell
ipssh_shell_stop_cb() – terminate the shell
ipssh_shell_welcome_msg_cb() – display welcome message at shell creation
ipssh_sock_shell_start_cb() – create shell with socket connection
ipssh_stop_all_spawns() – stop all spawned IPSSH tasks
ipssh_stop_spawn() – stop a spawned process
ipssh_validate_pubkey_cb() – validate client’s public key in SSH v2
ipssh_validate_rsa_pubkey_v1_cb() – validate client public key in SSH v1
ipssh_validate_userid_pw_cb() – validate user’s login ID and password
ipssh_configure()

NAME
ipssh_configure() – function to place any custom initialization code in.

SYNOPSIS
IP_GLOBAL Ip_err ipssh_configure(void);

DESCRIPTION
This function is called at boot, before the IPSSH daemon is started. Any custom configuration should be performed in this function.

Parameters:
None.

RETURNS
IPCOM_SUCCESS or IPCOM_ERR_FAILED Returning anything but IPCOM_SUCCESS will prevent the daemon from starting.

ERRNO

SEE ALSO
ipssh_config

ipssh_conn_new()

NAME
ipssh_conn_new() – create a new SSH connection handle

SYNOPSIS
IP_PUBLIC struct Ipssh_conn_st* ipssh_conn_new
    (void);

DESCRIPTION
This routine creates an SSH connection handle.

RETURNS
A connection handle or IP_NULL, if the operation fails.

ERRNO

SEE ALSO
ipssh

ipssh_evp_pkey_cmp()

NAME
ipssh_evp_pkey_cmp() – compare two public keys
SYNOPSIS

```
IP_PUBLIC int ipssh_evp_pkey_cmp
    (EVP_PKEY* key1,
     EVP_PKEY* key2);
```

DESCRIPTION

This routine compares two public keys. It is a utility function to aid implementers of `ipssh_validate_public_key`.

Parameters:

- `key1`:
  A pointer to the first public key to be used in the comparison.

- `key2`:
  A pointer to the second public key to be used in the comparison.

RETURNS

0, if keys are equal; non-zero, otherwise.

ERRNO

SEE ALSO

ipssh

---

**ipssh_get_clt_fd()**

NAME

`ipssh_get_clt_fd()` – return the socket descriptor of a client

SYNOPSIS

```
IP_PUBLIC int ipssh_get_clt_fd
    (void* ssh_ctx);
```

DESCRIPTION

This routine returns the socket descriptor of a client.

If IPSSH is configured to listen to several ports and each port is connected to a different shell, it is important to be able to determine what port a client connected on in the shell start callback routine, so that the start callback can initiate the correct shell for each connection. This routine returns the socket descriptor, which in turn is used in a call to `ipcom_getpeername()` to get the port the client connected on.

Parameter:

- `ssh_ctx`:
  A handle to the SSH connection context.

RETURNS

The socket descriptor of the client connection.
ipssh_get_clt_info()

NAME
ipssh_get_clt_info() – return information on logged-in client

SYNOPSIS
IP_PUBLIC Ip_err ipssh_get_clt_info
(
   Ipssh_clt_info* clt_info,
   int pos
);

DESCRIPTION
This routine returns information on logged-in client.

Parameters:
clt_info
   A pointer to an Ipssh_clt_info structure.
pos
   An index value indicating the client for which information is being returned

RETURNS
IPCOM_SUCCESS, or IPCOM_ERR_NOT_FOUND, if pos exceeds the number of logged in clients.

ERRNO

SEE ALSO
ipssh

ipssh_get_sftp_func_cb()

NAME
ipssh_get_sftp_func_cb() – perform file operations on a specified file

SYNOPSIS
IP_PUBLIC Ipssh_sftp_func_cb* ipssh_get_sftp_file_func_cb
(
    void
);

DESCRIPTION
This callback routine is called once for each request for a file operation on a specific path. It returns a set of pointers to file-system routines that are called during the processing of the
request. Note that a typical file operation, such as a get or a put operation, consists of several SFTP messages, so that a single file operation request may result in several calls to the routine pointers.

Parameter:

path
The path to the file on which a file operation is to be performed.

RETURNS
A pointer to Ipssh_sftp_func_cb, which is defined as follows:

```c
struct Ipssh_sftp_func_cb_st{
    char type[16];
    /* File management callbacks */
    Ip_fd (*open_cb)(const char *pathname, int omode, Ip_mode_t attrs);
    Ip_err (*close_cb)(Ip_fd fd);
    Ip_err (*stat_cb)(const char* path, struct Ip_stat* buf);
    int (*read_cb)(Ip_fd fd, char* data, Ip_size_t len);
    int (*write_cb)(Ip_fd fd, const char* data, Ip_size_t len);
    int (*lseek_cb)(Ip_fd fd, Ip_u64 offset, int whence);
    /* Directory mgnt callbacks */
    IP_DIR* (*opendir_cb)(const char* path);
    int (*closedir_cb)(IP_DIR* dirp);
    struct Ip_dirent* (*readdir_cb)(IP_DIR *dirp);
    int (*mkdir_cb)(const char* path, Ip_mode_t mode);
    int (*rmdir_cb)(const char* path);
};
typedef struct Ipssh_sftp_func_cb_st Ipssh_sftp_func_cb;
```

ERRNO

SEE ALSO
ipssh

### ipssh_key_fingerprint()

**NAME**  
ipssh_key_fingerprint() – generate a fingerprint of a public key

**SYNOPSIS**  
IP_PUBLIC Ip_err ipssh_key_fingerprint
(  
    EVP_PKEY* key,
    char** fp,
    int fp_type
);

**DESCRIPTION**  
Generates a fingerprint of a public key.

Parameters:
ipssh_load_dsa_key( )

**NAME**

ipssh_load_dsa_key( ) – return the DSA server key to be used by the daemon

**SYNOPSIS**

```c
IP_PUBLIC DSA* ipssh_load_dsa_key(void);
```

**DESCRIPTION**

This function is called when the IPSSH daemon is started. The function must return the DSA server key to be used by the daemon.

Parameters:

None.

**RETURNS**

An RSA key or IP_NULL to indicate that no RSA key is available. Returning will prevent the daemon from starting.

**ERRNO**

SEE ALSO

ipssh

ipssh_load_rsa_key( )

**NAME**

ipssh_load_rsa_key( ) – return the RSA server key to be used by the daemon
SYNOPSIS
   IP_PUBLIC RSA* ipssh_load_rsa_key(void);

DESCRIPTION
This function is called when the IPSSH daemon is started. The function must return the RSA
default server key to be used by the daemon.

Parameters:
None.

RETURNS
An RSA key or IP_NULL to indicate that no RSA key is available. Returning IP_NULL will
prevent the daemon from starting.

ERRNO

SEE ALSO
ipssh_config

ipssh_send Shell_stdout_data()

NAME
ipssh_send Shell_stdout_data() – send stdout data to the SSH client

SYNOPSIS
   IP_PUBLIC int ipssh_send_shell_stdout_data
   (  
      void* ssh_ctx,
      void* shell_ctx,
      const char* data,
      int len
   );

DESCRIPTION
Parameters:

ssh_ctx
   A handle to the ssh connection context.

shell_ctx
   Handle to the shell context. This handle is returned by the shell start callback and
   transparently conveyed by IPSSH.

data
   Pointer to the stdout data

len
   Length of the stdout data.

RETURNS
The number of bytes written or IPCOM_ERR_FAILED, if no bytes were written.
ipssh_sftp_check_access_cb()

NAME
ipssh_sftp_check_access_cb() – check user's access to file system

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_check_access_cb
(    const char* user,
    int         action,
    const char* path,
    int         flags);

DESCRIPTION
This callback routine checks whether the user is authorized to perform a file operation. It is called whenever the user initiates a file operation.

Parameters:
user
    ID of the user initiating the request.
action
    The file operation requested. The following operations are possible:
    IPSSH_FXP_MKDIR()
        Create a directory.
    IPSSH_FXP_RMDIR()
        Delete a directory.
    IPSSH_FXP_REMOVE()
        Delete a file.
    IPSSH_FXP_OPEN()
        Read a file or write to a file, as indicated by the flag parameter.
path
    The path to the file.
flags
    One of the following flags for opening a file (used only with the IPSSH_FXP_OPEN file operation):
    IPSSH_FXP_READ
        Open the file for reading.
**IPSSH_FXP_WRITE**
Open the file for writing.

**RETURNS**
Either **IPCOM_SUCCESS** or an error code (see **ipcom_err.h**).

**ERRNO**

**SEE ALSO**
ipssh_config

---

**ipssh_sftp_clt_chdir()**

**NAME**
ipssh_sftp_clt_chdir() – change current working directory on the server

**SYNOPSIS**

```c
IP_PUBLIC Ip_err ipssh_sftp_clt_chdir(
    struct Ipssh_conn_st* ssh_conn,
    const char*           path
);
```

**DESCRIPTION**
This routine changes the current working directory on the SFTP server.

**Parameters:**
- `ssh_conn`  
  A handle to the connection.
- `path`  
  The path to the new working directory, either relative or absolute.

**RETURNS**
- **IPCOM_SUCCESS** or the following error:
  - **IPCOM_ERR_FAILED**  
    Working directory not updated.

**ERRNO**

**SEE ALSO**
ipssh_sftp_clt

---

**ipssh_sftp_clt_close()**

**NAME**
ipssh_sftp_clt_close() – close a connection
ipssh_sftp_clt_closedir()

**SYNOPSIS**

```c
IP_PUBLIC Ip_err ipssh_sftp_clt_close
{
    struct Ipssh_conn_st* ssh_conn;
};
```

**DESCRIPTION**

Disconnects an SFTP connection and frees all resources tied to the connection.

Parameters:

- `ssh_conn`
  A handle to the connection.

**RETURNS**

- IPCOM_SUCCESS or an error code (see ipcom_err.h).

**SEE ALSO**

ipssh_sftp_clt

ipssh_sftp_clt_closedir()

**NAME**

`ipssh_sftp_clt_closedir()` – close a directory stream on the server

**SYNOPSIS**

```c
IP_PUBLIC Ip_err ipssh_sftp_clt_closedir
{
    struct Ipssh_conn_st* ssh_conn,
    void*                 handle
};
```

**DESCRIPTION**

This routine closes a directory stream that was opened with `ipssh_sftp_clt_opendir()`.

Parameters:

- `ssh_conn`
  A handle to the connection.

- `handle`
  A handle to the directory stream to be closed.

**RETURNS**

- IPCOM_SUCCESS or the following error:
  - IPCOM_ERR_FAILED
    Communication with the SFTP server failed.

**SEE ALSO**

ipssh_sftp_clt
ipssh_sftp_clt_get()

NAME
ipssh_sftp_clt_get() – get a file from the server

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_clt_get
    (struct Ipssh_conn_st* ssh_conn,
     const char*           src_path,
     const char*           dst_path
    );

DESCRIPTION
This routine transfers a file from the SFTP server to the local file system.

Parameters:
ssh_conn
    A handle to the connection.
src_path
    The path to the source file on the SFTP server.
dst_path
    The Path to the destination file on the local file system.

RETURNS
IPCOM_SUCCESS or one of the following errors:
IPCOM_ERR_NOT_FOUND
    The file does not exist on the SFTP server.
IPCOM_ERR_PERMISSION_DENIED
    Insufficient permission to access file on SFTP server.
IPCOM_ERR_FAILED
    Operation failed.

ERRNO

SEE ALSO
ipssh_sftp_clt

ipssh_sftp_clt_get_cwd()

NAME
ipssh_sftp_clt_get_cwd() – get current working directory on the server

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_clt_get_cwd
    (struct Ipssh_conn_st* ssh_conn,
     const char*           src_path,
     const char*           dst_path
    );
ipssh_sftp_clt_get realpath()

char* buf,
int buf_len

DESCRIPTION
This routine returns the current working directory on the SFTP server.
Parameters:
ssh_conn
A handle to the connection.
buf
The buffer for holding the returned directory name.
buf_len
The length of the buffer, in bytes.
RETURNS
IPCOM_SUCCESS or one of the following errors:
IPCOM_ERR_INVALID_ARGS
Buffer is too small to contain the returned directory name.
IPCOM_ERR_FAILED
Communication with server failed.
ERRNO
SEE ALSO
ipssh_sftp_clt

ipssh_sftp_clt_get realpath()

IP_PUBLIC Ip_err ipssh_sftp_clt_get realpath
(
    struct Ipssh_conn_st* ssh_conn,
    const char* path,
    char* buf,
    int buf_len
);

DESCRIPTION
This routine converts a relative path on the server into an absolute path. This is useful for
converting a path containing .. or a relative pathname without a leading slash into an
absolute path.
Parameters:
**ipssh_sftp_clt_mkdir( )**

**NAME**

`ipssh_sftp_clt_mkdir( )` – create a directory on the server

**SYNOPSIS**

```c
IP_PUBLIC Ip_err ipssh_sftp_clt_mkdir(
   struct Ipssh_conn_st* ssh_conn,
   const char*           path,
   Ip_mode_t             mode
);
```

**DESCRIPTION**

This routine creates a directory on the remote server.

Parameters:

- `ssh_conn`  
  A handle to the connection.

- `path`  
  The path to the directory to be created.

- `mode`  
  An integer specifying the file privileges that apply to the directory. The integer is usually expressed as an octal value, read (4), write (2) and execute (1). These values can...
be combined, e.g., 6 means read and write permission. Three octal values are used to express the privileges for the user, group, and others, respectively. For example, 0644 means the user has read and write privileges, the group has read privileges, and others have read privileges.

**RETURNS**

IPCOM_SUCCESS if the directory was created.

**ERRNO**


**SEE ALSO**

ipssh_sftp_clt

### ipssh_sftp_clt_open()

**NAME**

ipssh_sftp_clt_open() – open a connection to an SFTP server

**SYNOPSIS**

IP_PUBLIC Ip_err ipssh_sftp_clt_open

(struct Ipssh_conn_st* ssh_conn,
 const char* hostname,
 Ip_u16 port);

**DESCRIPTION**

This routine establishes a connection to an SFTP server.

Parameters:

- **ssh_conn**
  - Connection handle.
- **hostname**
  - Host name or IP address of the SFTP server.
- **port**
  - Port on which the SFTP server listens.

**RETURNS**

IPCOM_SUCCESS or one of the following errors:

- **IPCOM_ERR_INVALID_ARG**
  - Arguments are bad; for example, no authentication data set.
- **IPCOM_ERR_AUTH_FAILED**
  - Server rejected the login request.
- **IPCOM_ERR_FAILED**
  - Connection could not be established.
NAME

ipssh_sftp_clt_opendir() – open a directory stream on a server

SYNOPSIS

IP_PUBLIC void* ipssh_sftp_clt_opendir

(struct Ipssh_conn_st* ssh_conn,
 const char* path);

DESCRIPTION

This routine opens a directory stream for a specified directory. The stream is positioned at the first entry in the directory.

Parameters:

ssh_conn
A handle to the connection.

path
The path to the directory.

RETURNS

A pointer to the directory stream, or IP_NULL if the operation failed.

ERRNO

SEE ALSO

ipssh_sftp_clt

NAME

ipssh_sftp_clt_put() – transfer a file to the server

SYNOPSIS

IP_PUBLIC Ip_err ipssh_sftp_clt_put

(struct Ipssh_conn_st* ssh_conn,
 const char* src_path,
 const char* dst_path);

DESCRIPTION

This routine transfers a file from the local file system to the SFTP server.
Parameters:

ssh_conn
A handle to the connection.
	src_path
The path to the source file on the local file system.
	dst_path
The path to the destination file on the SFTP server.

RETURNS
IPCOM_SUCCESS or one of the following errors:

IPCOM_ERR_NOT_FOUND
File does not exist on the SFTP server.

IPCOM_ERR_PERMISSION_DENIED
Insufficient permission to access file on SFTP server.

IPCOM_ERR_FAILED
Operation failed.

ERRNO
SEE ALSO
ipssh_sftp_clt

ipssh_sftp_clt_readdir( )

NAME
ipssh_sftp_clt_readdir( ) – read the next directory entry on the server

SYNOPSIS
IP_PUBLIC Ipssh_dirent* ipssh_sftp_clt_readdir
    (struct Ipssh_conn_st* ssh_conn,
     void* handle);

DESCRIPTION
This routine returns a pointer to an ipssh_dirent structure containing the next directory entry in the specified directory stream.

Parameters:

ssh_conn
A handle to the connection.

handle
A handle to the directory stream.

RETURNS
A pointer to an ipssh_dirent structure, or IP_NULL if the last entry has been read.
**NAME**

`ipssh_sftp_clt_rename()` – rename a file on the server

**SYNOPSIS**

```c
IP_PUBLIC int ipssh_sftp_clt_rename
    (struct Ipssh_conn_st* ssh_conn,
     const char*           oldpath,
     const char*           newpath);
```

**DESCRIPTION**

This routine renames a file on the server.

Parameters:

- `ssh_conn`  
  A handle to the connection.

- `oldpath`  
  path to the file to be renamed

- `newpath`  
  new path for the renamed file

**RETURNS**

- `IPCOM_SUCCESS` or one of the following errors:
  - `IPCOM_ERR_NOT_FOUND`
    File not found on SFTP server.
  - `IPCOM_ERR_PERMISSION_DENIED`
    Insufficient privileges to rename file.
  - `IPCOM_ERR_FAILED`
    Communication with SFTP server failed.

**ERRNO**

SEE ALSO

`ipssh_sftp_clt`
ipssh_sftp_clt_rmdir()

NAME

ipssh_sftp_clt_rmdir() – delete a directory on the server

SYNOPSIS

IP_PUBLIC Ip_err ipssh_sftp_clt_rmdir
{
  struct Ipssh_conn_st* ssh_conn,
  const char*           path
};

DESCRIPTION

The routine deletes a directory on the remote server.

Parameters:

  ssh_conn
    A handle to the connection.

  path
    The path to the directory to be deleted.

RETURNS

IPCOM_SUCCESS, if the directory was deleted. [* What if not deleted--e.g., insufficient privilege]

ERRORS

SEE ALSO

ipssh_sftp_clt

ipssh_sftp_clt_set_key()

NAME

ipssh_sftp_clt_set_key() – set key pair for authentication

SYNOPSIS

IP_PUBLIC Ip_err ipssh_sftp_clt_set_key
{
  struct Ipssh_conn_st* ssh_conn,
  const EVP_PKEY*       key
};

DESCRIPTION

This routine sets the key pair that will be used for authenticating to the SFTP server. The reference counter of the key is incremented. Only DSA key pairs are currently supported.

Parameters:

  key
    Key pair.
ipssh_sftp_clt_set_pw()

NAME
ipssh_sftp_clt_set_pw() – set password for authentication

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_clt_set_pw
  (  
    struct Ipssh_conn_st* ssh_conn,
    const char* pw
  );

DESCRIPTION
This routine sets the password to be used for authenticating to the SFTP server.

Parameters:
  
pw
    The password to be used for authentication.

RETURNS
IPCOM_SUCCESS or the following error:
IPCOM_ERR_INVALID_ARG
  Password is too long.

ERRNO

SEE ALSO
ipssh_sftp_clt

ipssh_sftp_clt_set_userid()

NAME
ipssh_sftp_clt_set_userid() – set user ID for authentication

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_clt_set_userid
  (  
    struct Ipssh_conn_st* ssh_conn,
    const char* user
  );
DESCRIPTION
This routine sets the user ID to be used for authenticating to the SFTP server.

Parameters:
user
User ID to be used for authentication.

RETURNS
IPCOM_SUCCESS or the following error:
IPCOM_ERR_INVALID_ARG
User ID is too long.

ERRNO
SEE ALSO
ipssh_sftp_clt

---

ipssh_sftp_clt_unlink()

NAME
ipssh_sftp_clt_unlink() – delete a file on the server

SYNOPSIS
IP_PUBLIC Ip_err ipssh_sftp_clt_unlink
    (const Ipssh_conn_st* ssh_conn,
     const char* path);

DESCRIPTION
This routine deletes a file from the server.

Parameters:
ssh_conn
A handle to the connection.

RETURNS
IPCOM_SUCCESS or one of the following errors:
IPCOM_ERR_NOT_FOUND
File not found on SFTP server.
IPCOM_ERR_PERMISSION_DENIED
Insufficient privileges to remove file.
IPCOM_ERR_FAILED
Communication with SFTP server failed.

ERRNO
SEE ALSO
ipssh_sftp_clt
ipssh_shell_exit()

NAME
ipssh_shell_exit() – report that a shell has terminated

SYNOPSIS
IP_PUBLIC Ip_err ipssh_shell_exit
{
    void* ssh_ctx,
    void* shell_ctx
};

DESCRIPTION
This routine informs IPSSH that a shell has terminated. If the shell is the last shell connected
to the SSH connection, the connection is released.

Parameters:

ssh_ctx
    A handle to the SSH connection context.

shell_ctx
    Handle to the shell context. This handle is returned by the shell start callback and
    transparently conveyed by IPSSH.

RETURNS
IPCOM_SUCCESS or an error code.

ERRNO

SEE ALSO
ipssh

ipssh_shell_start_cb()

NAME
ipssh_shell_start_cb() – start a shell at user login

SYNOPSIS
IP_PUBLIC void* ipssh_shell_start_cb
{
    void* ssh_ctx,
    const char* user,
    void* cookie
};

DESCRIPTION
This callback routine starts a shell when a user logs in. It returns a handle to the shell context,
which is used by ipssh_shell_stdin_cb() and ipssh_shell_stop_cb(). The SSH protocol
allows several shells to be started on the same connection. The IPCOM shell does not
support this capability, but SSH does. Hence, multiple shell on the same connection can only
be used if a proprietary shell is integrated.
Parameters:

ssh_ctx
A handle to the current SSH connection.

user
A user ID.

cookie
A pointer to the cookie used in authentication by either `ipssh_validate_pubkey_cb()` or `ipssh_validate_rsa_pubkey_v1_cb()`.

RETURNS
Handle to the shell context. A return value of IP_NULL releases the connection.

ERRNO

SEE ALSO
ipssh_config

### `ipssh_shell_stdin_cb()`

**NAME**
`ipssh_shell_stdin_cb()` – decrypt standard-input data and send to shell

**SYNOPSIS**
```c
IP_PUBLIC Ip_err ipssh_shell_stdin_cb
    (void* ssh_ctx,
     void* shell_ctx,
     const char* data,
     int len);
```

**DESCRIPTION**
This callback routine decrypts standard-input data and delivers it to the shell. It is called whenever data is received on the SSH connection.

**Parameters:**

*ssh_ctx*
A handle to the SSH connection context.

*shell_ctx*
A handle to the shell context. The handle was originally returned by `ipssh_shell_start_cb()`, and is transparently conveyed by SSH.

*data*
A pointer to standard-input data.

*len*
The length of the standard-input data.
**ipssh_shell_stop_cb( )**

**NAME**  
`ipssh_shell_stop_cb( )` – terminate the shell

**SYNOPSIS**  
```c
IP_PUBLIC Ip_err ipssh_shell_stop_cb( 
    void* ssh_ctx,  
    void* shell_ctx
);
```

**DESCRIPTION**  
This callback routine terminates a shell. It is called whenever a shell needs to terminate; for example, when the client disconnects.

Parameters:

- `ssh_ctx`  
  A handle to the SSH connection context.

- `shell_ctx`  
  A handle to the shell context, as returned by `ipssh_shell_start_cb( )` and transparently conveyed by SSH.

**RETURNS**  
Either `IPCOM_SUCCESS` or an error code (see `ipcom_err.h`).

**ERRNO**

**SEE ALSO**  
ipssh_config

**ipssh_shell_welcome_msg_cb( )**

**NAME**  
`ipssh_shell_welcome_msg_cb( )` – display welcome message at shell creation

**SYNOPSIS**  
```c
IP_PUBLIC Ip_err ipssh_shell_welcome_msg_cb( 
    void* ssh_ctx,
    void* shell_ctx,
    char* welcome_msg
);
```

**DESCRIPTION**  
This callback routine displays a welcome message at shell creation.

Parameters:

- `ssh_ctx`  
  A handle to the SSH connection context.

- `shell_ctx`  
  A handle to the shell context, as returned by `ipssh_shell_start_cb( )` and transparently conveyed by SSH.

- `welcome_msg`  
  A string containing the welcome message.

**RETURNS**  
Either `IPCOM_SUCCESS` or an error code (see `ipcom_err.h`).

**ERRNO**

**SEE ALSO**  
ipssh_config
DESCRIPTION
This callback routine outputs a welcome message or banner. It is called when a shell has been created and all SSH internal setup operations have been completed.

Parameters:

- `ssh_ctx`:
  A handle to the current SSH connection.

- `shell_ctx`:
  A handle to the current shell context.

RETURNS
Either `IPCOM_SUCCESS` or an error code (see `ipcom_err.h`). An error releases the SSH connection.

SEE ALSO
`ipssh_config`

---

**ipssh_sock_shell_start_cb()**

**NAME**
`ipssh_sock_shell_start_cb()` – create shell with socket connection

**SYNOPSIS**
```c
IP_PUBLIC Ip_err ipssh_sock_shell_start_cb
     (Ip_fd *stdio_sock,
      Ip_fd client_fd);
```

**DESCRIPTION**
This callback routine creates a shell and connects a TCP socket to the standard input and standard output of the shell.

Parameters:

- `stdio_sock`:
  A pointer to the standard-input file descriptor to be set.

- `client_fd`:
  The file descriptor for the client connection, which can be used to obtain the IP address of the client.

RETURNS
Either `IPCOM_SUCCESS` or an error code (see `ipcom_err.h`). An error releases the SSH connection.
ipssh_stop_all_spawns()

NAME    ipssh_stop_all_spawns() – stop all spawned IPSSH tasks

SYNOPSIS  IP_PUBLIC Ip_err ipssh_stop_all_spawns(void);

DESCRIPTION  This routine stops all spawned IPSSH tasks. This is equivalent to logging out all users.

Parameters:
None.

RETURNS  IPCOM_SUCCESS or IPCOM_ERR_FAILED

ERRNO    SEE ALSO  ipssh

ipssh_stop_spawn()

NAME    ipssh_stop_spawn() – stop a spawned process

SYNOPSIS  IP_PUBLIC Ip_err ipssh_stop_spawn
           (Ip_pid_t pid);

DESCRIPTION  This routine stops a spawned process.

Parameter:

    pid
    The process ID of the process to stop.

RETURNS  IPCOM_SUCCESS or IPCOM_ERR_NOT_INSTALLED, if there is no spawned process with the given process ID.
ipssh_validate_pubkey_cb()

NAME
ipssh_validate_pubkey_cb() – validate client's public key in SSH v2

SYNOPSIS
IP_PUBLIC Ip_err ipssh_validate_pubkey_cb
    (const char* user,
     EVP_PKEY*   pub_key,
     void**      cookie
    );

DESCRIPTION
This callback routine validates public keys used for client authentication in SSH v2
connections. The example configuration file ipssh/config/ipssh_config.'c' contains a sample
implementation.

Parameters:
userid
    The user ID sent by the client.
pub_key
    The public key used by the client for authentication.
cookie
    A pointer to the location of a cookie for use by ipssh_shell_start_cb().

RETURNS
Either IPCOM_SUCCESS, for a valid user ID and public key, or an error code (see
ipcom_err.h). An error causes SSH to reject the login attempt.

ERRNO

SEE ALSO
ipssh_config

ipssh_validate_rsa_pubkey_v1_cb()

NAME
ipssh_validate_rsa_pubkey_v1_cb() – validate client public key in SSH v1

SYNOPSIS
IP_PUBLIC RSA* ipssh_validate_rsa_pubkey_v1_cb
    (const char* user,
     EVP_PKEY*   pub_key,
     void**      cookie
    );

DESCRIPTION
This callback routine validates public keys used for client authentication in SSH v1
connections. The example configuration file ipssh/config/ipssh_config.'c' contains a sample
implementation.

Parameters:
userid
    The user ID sent by the client.
pub_key
    The public key used by the client for authentication.
cookie
    A pointer to the location of a cookie for use by ipssh_shell_start_cb().

RETURNS
Either IPCOM_SUCCESS, for a valid user ID and public key, or an error code (see
ipcom_err.h). An error causes SSH to reject the login attempt.

ERRNO

SEE ALSO
ipssh_config
This callback routine validates public RSA keys used for client authentication in SSH v1 connections. The configuration file `ipssh_config.c` contains a sample implementation.

Parameters:

- `userid`  
  The user ID sent by the client.

- `modulus`  
  The modulus of the key used by the client for authentication.

- `cookie`  
  A pointer to the location of a cookie for use by `ipssh_shell_start_cb()`.

Returns:

An RSA key structure containing the client's complete public key, both modulus (n) and public exponent (e). A return value of `IP_NULL` means that the public modulus does not match an authorized public key, in which case SSH rejects the login attempt.

Errno: `ipssh_config`

### ipssh_validate_userid_pw_cb()

**Name**  
ipssh_validate_userid_pw_cb() – validate user's login ID and password

**Synopsis**  

```c
IP_PUBLIC Ip_err ipssh_validate_userid_pw_cb
{
    const char* user,
    const char* pw,
    void**      cookie

    
}
```

**Description**  
This callback routine validates a user's login ID and password. The configuration file `ipssh/config/ipssh_config.c` contains a sample implementation.

Parameters:

- `user`  
  The user ID provided by the client.
**B Routines**

*ipssh_validate_userid_pw_cb ( )*

---

**pw**
The password provided by the client.

**cookie**
Pointer to a cookie for use by *ipssh_shell_start_cb ( ).*

---

**RETURNS**
IPCOM_SUCCESS for a valid user ID and password. All other return values cause SSH to reject the login attempt.

---

**ERRNO**

---

**SEE ALSO**
*ipssh_config*
Index

Numerics

3DES algorithm 11

A

access control callback routines 4
access control, for SFTP client 27
AES algorithm 11
algorithms, version 2.0 support 3
ARCFOUR algorithm 12
attempts, authentication 9
AUTH_USER_NAME1 19
AUTH_USER_PASSWD1 19
authentication 17
attempts 9
password 19
public key 19
server 18
user 19
authentication protocol, overview 2

B

Blowfish algorithm 11

C

C API, for shell integration 23
callback routines 17
access control 4
for file system 26
CAST algorithm 11
cd command 33
cipher, allowing none 12
client socket descriptor 25
client timeout 10
clients, maximum simultaneous 10
command-interpreter commands 29
commands
cd 33
exit 38
get 33
lcd 36
lls 37
lnkdr 37
lpwd 36
lrm 37
lrmdir 38
ls 35
put 34
pwd 35
quit 38
rm 35
SFTP client 32
ipssh.v2 21
ipssh_config.c 17, 22
ipssh_config.h 18
ipssh_configure() 17
Ipssh_conn_st struct 25
ipssh_get_clt_fd() 25
ipssh_get_sftp_func_cb() 26
ipssh_list command 30
ipssh_sftp_check_access_cb() 27
ipssh_sftp_clt_open() 26
ipssh_sftp_clt_set_key() 26
ipssh_sftp_clt_set_pw() 26
ipssh_sftp_clt_set_userid() 25
ipssh_sftp_file.c 26
ipssh_shell_start_cb() 22, 25
ipssh_stop command 30
IPSSH_USE_DEFAULT_KEYS 18
ipssh_validate_pubkey_cb() 19
ipssh_validate_userid_pw_cb() 19

K

kernel applications, developing 16
key fingerprint 31
keyfp command 31

L

lcd command 36
listing SSH clients 30
lls command 37
lmkdir command 37
lpwd command 36
lrn command 37
lrmindir command 38
ls command 35

M

MAC algorithm
MD5 11
MD5-96 12
SHA1 12
SHA1-96 12
MAC algorithm, disable 11
maximum shells 10
MD5 MAC algorithm 11
MD5-96 MAC algorithm 12
multiple ports 21, 25
multiple shells 25
multiple shells and ports, using 25

N

no cipher, enabling 12

O

overview
authentication protocol 2
connection protocol 2
SSH file transfer protocol 2
transport protocol 2

P

parameters, configuration 8
password authentication 19
  allowed 9
  required 9
password, setting for SFTP client 26
permissions, for files and directories 27
pk2 file, for user authentication 19
port forwarding 10
port number 11, 25
ports, multiple 21, 25
public key authentication 19
  allowed 9
  performed first 9
  required 10
public key, setting for SFTP client 26
put command 34
starting Wind River SSH 25
stopping SSH processes 30
stopping Wind River SSH 25
system variables (sysvars) 20
sysvars 20

T
TCP port name 11
telnet server 24
terminal connections 4
terminal service, enabling 10
timeout, client 10
transport module 3
transport protocol, overview 2

U
user authentication 19
user authentication module 3
userid, setting for SFTP client 25
utility commands 30

V
VxWorks components 7
VxWorks shell 22

W
Wind River documentation 5
working directory
  changing 33, 36
  printing 35, 36