rednet

Communicate with other computers by using **modems**. **`rednet`** provides a layer of abstraction on top of the main **`modem`** peripheral, making it slightly easier to use.

Basic usage

In order to send a message between two computers, each computer must have a modem on one of its sides (or in the case of pocket computers and turtles, the modem must be equipped as an upgrade). The two computers should then call **`rednet.open`**, which sets up the modems ready to send and receive messages.

Once rednet is opened, you can send messages using **`rednet.send`** and receive them using **`rednet.receive`**. It's also possible to send a message to *every* rednet-using computer using **`rednet.broadcast`**.

> ⚠ **NETWORK SECURITY**
>
> While rednet provides a friendly way to send messages to specific computers, it doesn't provide any guarantees about security. Other computers could be listening in to your messages, or even pretending to send messages from other computers!
>
> If you're playing on a multi-player server (or at least one where you don't trust other players), it's worth encrypting or signing your rednet messages.

Protocols and hostnames

Several rednet messages accept "protocol"s - simple string names describing what a message is about. When sending messages using **`rednet.send`** and **`rednet.broadcast`**, you can optionally specify a protocol for the message. This same protocol can then be given to **`rednet.receive`**, to ignore all messages not using this protocol.

It's also possible to look-up computers based on protocols, providing a basic system for service discovery and **DNS**. A computer can advertise that it supports a particular protocol with **`rednet.host`**, also providing a friendly "hostname". Other computers may then find all computers which support this protocol using **`rednet.lookup`**.

See also

- `rednet_message` Queued when a rednet message is received.
- `modem` Rednet is built on top of the modem peripheral. Modems provide a more bare-bones but flexible interface.

Changes

- **New in version 1.2**

`CHANNEL_BROADCAST = 65535`	The channel used by the Rednet API to `broadcast` messages.
`CHANNEL_REPEAT = 65533`	The channel used by the Rednet API to repeat messages.
`MAX_ID_CHANNELS = 65500`	The number of channels rednet reserves for computer IDs.
`open(modem)`	Opens a modem with the given `peripheral` name, allowing it to send and receive messages over rednet.
`close([modem])`	Close a modem with the given `peripheral` name, meaning it can no longer send and receive rednet messages.
`isOpen([modem])`	Determine if rednet is currently open.
`send(recipient, message [, protocol])`	Allows a computer or turtle with an attached modem to send a message intended for a computer with a specific ID.
`broadcast(message [, protocol])`	Broadcasts a string message over the predefined `CHANNEL_BROADCAST` channel.
`receive([protocol_filter [, timeout]])`	Wait for a rednet message to be received, or until `nTimeout` seconds have elapsed.
`host(protocol, hostname)`	Register the system as "hosting" the desired protocol under the specified name.
`unhost(protocol)`	Stop **hosting** a specific protocol, meaning it will no longer respond to `rednet.lookup` requests.

`lookup(protocol [, hostname])`	Search the local rednet network for systems **hosting** the desired protocol and returns any computer IDs that respond as "r...
`run()`	Listen for modem messages and converts them into rednet messages, which may then be **received**.

§ `CHANNEL_BROADCAST = 65535` [Source]

The channel used by the Rednet API to **`broadcast`** messages.

§ `CHANNEL_REPEAT = 65533` [Source]

The channel used by the Rednet API to repeat messages.

§ `MAX_ID_CHANNELS = 65500` [Source]

The number of channels rednet reserves for computer IDs. Computers with IDs greater or equal to this limit wrap around to 0.

§ `open(modem)` [Source]

Opens a modem with the given **`peripheral`** name, allowing it to send and receive messages over rednet.

This will open the modem on two channels: one which has the same **ID** as the computer, and another on **the broadcast channel**.

Parameters

1. `modem` : **`string`** The name of the modem to open.

Throws

- If there is no such modem with the given name

Usage

- Open rednet on the back of the computer, allowing you to send and receive rednet messages using it.

```
rednet.open("back")
```

Run ▷

- Open rednet on all attached modems. This abuses the "filter" argument to **`peripheral.find`**.

  ```
  peripheral.find("modem", rednet.open)
  ```
  Run ▷

See also

- **`rednet.close`**
- **`rednet.isOpen`**

---

§ `close([modem])`

[Source]

Close a modem with the given **`peripheral`** name, meaning it can no longer send and receive rednet messages.

Parameters

1. `modem?` : **`string`** The side the modem exists on. If not given, all open modems will be closed.

Throws

- If there is no such modem with the given name

See also

- **`rednet.open`**

---

§ `isOpen([modem])`

[Source]

Determine if rednet is currently open.

Parameters

1. `modem?` : **`string`** Which modem to check. If not given, all connected modems will be checked.

Returns

1. `boolean` If the given modem is open.

See also

- **`rednet.open`**

Changes

- **New in version 1.31**

§ `send(recipient, message [, protocol])` [Source]

Allows a computer or turtle with an attached modem to send a message intended for a computer with a specific ID. At least one such modem must first be **opened** before sending is possible.

Assuming the target was in range and also had a correctly opened modem, the target computer may then use **`rednet.receive`** to collect the message.

Parameters

1. `recipient` : `number` The ID of the receiving computer.
2. `message` : The message to send. Like with **`modem.transmit`**, this can contain any primitive type (numbers, booleans and strings) as well as tables. Other types (like functions), as well as metatables, will not be transmitted.
3. `protocol?` : **`string`** The "protocol" to send this message under. When using **`rednet.receive`** one can filter to only receive messages sent under a particular protocol.

Returns

1. `boolean` If this message was successfully sent (i.e. if rednet is currently **open**). Note, this does not guarantee the message was actually *received*.

Usage

- Send a message to computer #2.

```
rednet.send(2, "Hello from rednet!")
```
Run ▷

See also

- **`rednet.receive`**

Changes

- **Changed in version 1.6:** Added protocol parameter.
- **Changed in version 1.82.0:** Now returns whether the message was successfully sent.

§ `broadcast(message [, protocol])` [Source]

Broadcasts a string message over the predefined **`CHANNEL_BROADCAST`** channel. The message will be received by every device listening to rednet.

Parameters

1. `message` : The message to send. This should not contain coroutines or functions, as they will be converted to `nil`.
2. `protocol?` : `string` The "protocol" to send this message under. When using `rednet.receive` one can filter to only receive messages sent under a particular protocol.

Usage

- Broadcast the words "Hello, world!" to every computer using rednet.

```
rednet.broadcast("Hello, world!")
```

Run ▷

See also

- `rednet.receive`

Changes

- **Changed in version 1.6:** Added protocol parameter.

---

§ `receive([protocol_filter [, timeout]])`

[Source]

Wait for a rednet message to be received, or until `nTimeout` seconds have elapsed.

Parameters

1. `protocol_filter?` : `string` The protocol the received message must be sent with. If specified, any messages not sent under this protocol will be discarded.
2. `timeout?` : `number` The number of seconds to wait if no message is received.

Returns

1. `number` The computer which sent this message
2. The received message
3. `string | nil` The protocol this message was sent under.

Or

1. `nil` If the timeout elapsed and no message was received.

Usage

- Receive a rednet message.

```
local id, message = rednet.receive()
print(("Computer %d sent message %s"):format(id, message))
```

Run ▷

- Receive a message, stopping after 5 seconds if no message was received.

```lua
local id, message = rednet.receive(nil, 5)
if not id then
    printError("No message received")
else
    print(("Computer %d sent message %s"):format(id, message))
end
```
Run ▷

- Receive a message from computer #2.

```lua
local id, message
repeat
    id, message = rednet.receive()
until id == 2

print(message)
```
Run ▷

See also

- `rednet.broadcast`
- `rednet.send`

Changes

- **Changed in version 1.6:** Added protocol filter parameter.

---

§ `host(protocol, hostname)`

[Source]

Register the system as "hosting" the desired protocol under the specified name. If a rednet **lookup** is performed for that protocol (and maybe name) on the same network, the registered system will automatically respond via a background process, hence providing the system performing the lookup with its ID number.

Multiple computers may not register themselves on the same network as having the same names against the same protocols, and the title `localhost` is specifically reserved. They may, however, share names as long as their hosted protocols are different, or if they only join a given network after "registering" themselves before doing so (eg while offline or part of a different network).

Parameters

1. `protocol` : `string` The protocol this computer provides.
2. `hostname` : `string` The name this computer exposes for the given protocol.

Throws

- If trying to register a hostname which is reserved, or currently in use.

See also

- `rednet.unhost`
- `rednet.lookup`

Changes

- **New in version 1.6**

---

§ `unhost(protocol)`

[Source]

Stop **hosting** a specific protocol, meaning it will no longer respond to `rednet.lookup` requests.

Parameters

1. `protocol` : `string` The protocol to unregister your self from.

Changes

- **New in version 1.6**

---

§ `lookup(protocol [, hostname])`

[Source]

Search the local rednet network for systems **hosting** the desired protocol and returns any computer IDs that respond as "registered" against it.

If a hostname is specified, only one ID will be returned (assuming an exact match is found).

Parameters

1. `protocol` : `string` The protocol to search for.
2. `hostname?` : `string` The hostname to search for.

Returns

1. `number...` A list of computer IDs hosting the given protocol.

Or

1. `number | nil` The computer ID with the provided hostname and protocol, or `nil` if none exists.

Usage

- Find all computers which are hosting the `"chat"` protocol.

```lua
local computers = {rednet.lookup("chat")}
print(#computers .. " computers available to chat")
```

Run ▷

```lua
for _, computer in pairs(computers) do
  print("Computer #" .. computer)
end
```

- Find a computer hosting the `"chat"` protocol with a hostname of `"my_host"`.

  ```lua
  local id = rednet.lookup("chat", "my_host")
  if id then
    print("Found my_host at computer #" .. id)
  else
    printError("Cannot find my_host")
  end
  ```

  Run ▷

Changes

- **New in version 1.6**

---

§ run()

[Source]

Listen for modem messages and converts them into rednet messages, which may then be **received**.

This is automatically started in the background on computer startup, and should not be called manually.

Last updated on 2025-07-06