Examples: Server programs that use UNIX domain sockets

The following code example shows a server program for connection-oriented, stream UNIX domain sockets. In the server program, the socket function creates a stream socket in the UNIX domain, and then the bind function assigns a unique name for the socket. The listen function then accepts incoming client connections and creates a connection queue for further incoming requests.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a stream socket server sample program for UNIX \*/

/\* domain sockets. This program listens for a connection \*/

/\* from a client program, accepts it, reads data from the \*/

/\* client, then sends data back to connected UNIX socket. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#define SOCK\_PATH "tpf\_unix\_sock.server"

#define DATA "Hello from server"

int main(void){

int server\_sock, client\_sock, len, rc;

int bytes\_rec = 0;

struct sockaddr\_un server\_sockaddr;

struct sockaddr\_un client\_sockaddr;

char buf[256];

int backlog = 10;

memset(&server\_sockaddr, 0, sizeof(struct sockaddr\_un));

memset(&client\_sockaddr, 0, sizeof(struct sockaddr\_un));

memset(buf, 0, 256);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Create a UNIX domain stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_sock = socket(AF\_UNIX, SOCK\_STREAM, 0);

if (server\_sock == -1){

printf("SOCKET ERROR: %d\n", sock\_errno());

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set up the UNIX sockaddr structure \*/

/\* by using AF\_UNIX for the family and \*/

/\* giving it a filepath to bind to. \*/

/\* \*/

/\* Unlink the file so the bind will \*/

/\* succeed, then bind to that file. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_sockaddr.sun\_family = AF\_UNIX;

strcpy(server\_sockaddr.sun\_path, SOCK\_PATH);

len = sizeof(server\_sockaddr);

unlink(SOCK\_PATH);

rc = bind(server\_sock, (struct sockaddr \*) &server\_sockaddr, len);

if (rc == -1){

printf("BIND ERROR: %d\n", sock\_errno());

close(server\_sock);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Listen for any client sockets \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = listen(server\_sock, backlog);

if (rc == -1){

printf("LISTEN ERROR: %d\n", sock\_errno());

close(server\_sock);

exit(1);

}

printf("socket listening...\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Accept an incoming connection \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_sock = accept(server\_sock, (struct sockaddr \*) &client\_sockaddr, &len);

if (client\_sock == -1){

printf("ACCEPT ERROR: %d\n", sock\_errno());

close(server\_sock);

close(client\_sock);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Get the name of the connected socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

len = sizeof(client\_sockaddr);

rc = getpeername(client\_sock, (struct sockaddr \*) &client\_sockaddr, &len);

if (rc == -1){

printf("GETPEERNAME ERROR: %d\n", sock\_errno());

close(server\_sock);

close(client\_sock);

exit(1);

}

else {

printf("Client socket filepath: %s\n", client\_sockaddr.sun\_path);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Read and print the data \*/

/\* incoming on the connected socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("waiting to read...\n");

bytes\_rec = recv(client\_sock, buf, sizeof(buf), 0);

if (bytes\_rec == -1){

printf("RECV ERROR: %d\n", sock\_errno());

close(server\_sock);

close(client\_sock);

exit(1);

}

else {

printf("DATA RECEIVED = %s\n", buf);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Send data back to the connected socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

memset(buf, 0, 256);

strcpy(buf, DATA);

printf("Sending data...\n");

rc = send(client\_sock, buf, strlen(buf), 0);

if (rc == -1) {

printf("SEND ERROR: %d", sock\_errno());

close(server\_sock);

close(client\_sock);

exit(1);

}

else {

printf("Data sent!\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Close the sockets and exit \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

close(server\_sock);

close(client\_sock);

return 0;

}

The following code example shows a server program for connectionless, datagram UNIX domain sockets. In the server program, the socket function creates a datagram socket in the UNIX domain, and then the bind function assigns a unique name for the socket. For connectionless sockets, you do not have to issue the listen function to accept incoming requests from a client.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a datagram socket server sample program for UNIX \*/

/\* domain sockets. This program creates a socket and \*/

/\* receives data from a client. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#define SOCK\_PATH "tpf\_unix\_sock.server"

int main(void){

int server\_sock, len, rc;

int bytes\_rec = 0;

struct sockaddr\_un server\_sockaddr, peer\_sock;

char buf[256];

memset(&server\_sockaddr, 0, sizeof(struct sockaddr\_un));

memset(buf, 0, 256);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Create a UNIX domain datagram socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (server\_sock == -1){

printf("SOCKET ERROR = %d", sock\_errno());

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set up the UNIX sockaddr structure \*/

/\* by using AF\_UNIX for the family and \*/

/\* giving it a filepath to bind to. \*/

/\* \*/

/\* Unlink the file so the bind will \*/

/\* succeed, then bind to that file. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_sockaddr.sun\_family = AF\_UNIX;

strcpy(server\_sockaddr.sun\_path, SOCK\_PATH);

len = sizeof(server\_sockaddr);

unlink(SOCK\_PATH);

rc = bind(server\_sock, (struct sockaddr \*) &server\_sockaddr, len);

if (rc == -1){

printf("BIND ERROR = %d", sock\_errno());

close(server\_sock);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Read data on the server from clients \*/

/\* and print the data that was read. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("waiting to recvfrom...\n");

bytes\_rec = recvfrom(server\_sock, buf, 256, 0, (struct sockaddr \*) &peer\_sock, &len);

if (bytes\_rec == -1){

printf("RECVFROM ERROR = %d", sock\_errno());

close(server\_sock);

exit(1);

}

else {

printf("DATA RECEIVED = %s\n", buf);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Close the socket and exit \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

close(server\_sock);

return 0;

}

Examples: Client programs that use UNIX domain sockets

The following code example shows a client program for connection-oriented, stream UNIX domain sockets. In the program, the socket function is called to create a socket in the UNIX domain, and then the bind function assigns a unique name for the socket. The connect function then establishes a connection to the server.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a stream socket client sample program for UNIX \*/

/\* domain sockets. This program creates a socket, connects \*/

/\* to a server, sends data, then receives and prints a \*/

/\* message from the server. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#define SERVER\_PATH "tpf\_unix\_sock.server"

#define CLIENT\_PATH "tpf\_unix\_sock.client"

#define DATA "Hello from client"

int main(void){

int client\_sock, rc, len;

struct sockaddr\_un server\_sockaddr;

struct sockaddr\_un client\_sockaddr;

char buf[256];

memset(&server\_sockaddr, 0, sizeof(struct sockaddr\_un));

memset(&client\_sockaddr, 0, sizeof(struct sockaddr\_un));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Create a UNIX domain stream socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_sock = socket(AF\_UNIX, SOCK\_STREAM, 0);

if (client\_sock == -1) {

printf("SOCKET ERROR = %d\n", sock\_errno());

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set up the UNIX sockaddr structure \*/

/\* by using AF\_UNIX for the family and \*/

/\* giving it a filepath to bind to. \*/

/\* \*/

/\* Unlink the file so the bind will \*/

/\* succeed, then bind to that file. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_sockaddr.sun\_family = AF\_UNIX;

strcpy(client\_sockaddr.sun\_path, CLIENT\_PATH);

len = sizeof(client\_sockaddr);

unlink(CLIENT\_PATH);

rc = bind(client\_sock, (struct sockaddr \*) &client\_sockaddr, len);

if (rc == -1){

printf("BIND ERROR: %d\n", sock\_errno());

close(client\_sock);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set up the UNIX sockaddr structure \*/

/\* for the server socket and connect \*/

/\* to it. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_sockaddr.sun\_family = AF\_UNIX;

strcpy(server\_sockaddr.sun\_path, SERVER\_PATH);

rc = connect(client\_sock, (struct sockaddr \*) &server\_sockaddr, len);

if(rc == -1){

printf("CONNECT ERROR = %d\n", sock\_errno());

close(client\_sock);

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Copy the data to the buffer and \*/

/\* send it to the server socket. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

strcpy(buf, DATA);

printf("Sending data...\n");

rc = send(client\_sock, buf, strlen(buf), 0);

if (rc == -1) {

printf("SEND ERROR = %d\n", sock\_errno());

close(client\_sock);

exit(1);

}

else {

printf("Data sent!\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Read the data sent from the server \*/

/\* and print it. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("Waiting to recieve data...\n");

memset(buf, 0, sizeof(buf));

rc = recv(client\_sock, buf, sizeof(buf));

if (rc == -1) {

printf("RECV ERROR = %d\n", sock\_errno());

close(client\_sock);

exit(1);

}

else {

printf("DATA RECEIVED = %s\n", buf);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Close the socket and exit. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

close(client\_sock);

return 0;

}

The following code example shows a client program for connectionless, datagram UNIX domain sockets. In the program, the socket function is called to create a socket in the UNIX domain, and then the bind function assigns a unique name for the socket. For connectionless sockets, you do not have to issue the connect function to connect to the server.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This is a datagram socket client sample program for UNIX \*/

/\* domain sockets. This program creates a socket and sends \*/

/\* data to a server. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#define SERVER\_PATH "tpf\_unix\_sock.server"

#define DATA "Hello from client\n"

int main(void)

{

int client\_socket, rc;

struct sockaddr\_un remote;

char buf[256];

memset(&remote, 0, sizeof(struct sockaddr\_un));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Create a UNIX domain datagram socket \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_socket = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

printf("SOCKET ERROR = %d\n", sock\_errno());

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set up the UNIX sockaddr structure \*/

/\* by using AF\_UNIX for the family and \*/

/\* giving it a filepath to send to. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

remote.sun\_family = AF\_UNIX;

strcpy(remote.sun\_path, SERVER\_PATH);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Copy the data to be sent to the \*/

/\* buffer and send it to the server. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

strcpy(buf, DATA);

printf("Sending data...\n");

rc = sendto(client\_socket, buf, strlen(buf), 0, (struct sockaddr \*) &remote, sizeof(remote));

if (rc == -1) {

printf("SENDTO ERROR = %d\n", sock\_errno());

close(client\_sock);

exit(1);

}

else {

printf("Data sent!\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Close the socket and exit \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rc = close(client\_sock);

return 0;

}

* [**Create a pair of UNIX domain sockets by using the socketpair API**](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc1/unixsock_skpr.html)  
  You can use the socketpair function to create a pair of connected UNIX domain sockets. The pair of sockets is unnamed; that is, they are not bound to a file path.
* [**Display UNIX domain sockets**](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc1/unixsock_disp.html)  
  To display the active UNIX domain sockets that are on your system, use the **ZDTCP** command with the **NETSTAT** parameter specified.

**Parent topic:**

[Socket application design considerations](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc1/stackdc.html)

**Related reference**:

[bind: Bind a local name to the socket](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_bind.html#cpp_bind)

[socket: Create an endpoint for communication](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_socket.html#cpp_socket)

[socketpair: Create a pair of connected sockets](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_socketpair.html#cpp_socketpair)

[accept — Accept a connection request](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_accept.html)

[getpeername: Return the name of the peer](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_getpeername.html)

[listen — Complete binding, create connection request queue](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_listen.html)

[recv: Receive data on a connected socket](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_recv.html)

[recvfrom: Receive data on connected/unconnected socket](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_recvfrom.html)

[send: Send data on a connected socket](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtpc2/cpp_send.html)

**Related information**:

[Modifying existing applications to use SSL](https://www.ibm.com/docs/en/SSB23S_1.1.0.15/gtps7/s5mod.html)

While IBM values the use of inclusive language, terms that are outside of IBM's direct influence, for the sake of maintaining user understanding, are sometimes required. As other industry leaders join IBM in embracing the use of inclusive language, IBM will continue to update the documentation to reflect those changes.

© Copyright IBM Corporation 1993, 2019

* [Contact IBM](https://www.ibm.com/contact/us/en/?lnk=flg-cont-usen)
* [Privacy](https://www.ibm.com/privacy?lnk=flg-priv-usen)
* [Terms of use](https://www.ibm.com/legal?lnk=flg-tous-usen)
* [Accessibility](https://www.ibm.com/able/?lnk=flg-acce-usen)
* Cookie Preferences



* [IBM Documentation H](https://www.ibm.com/docs/en/about)