emVNC

CPU independent VNC stack
for embedded applications

User Guide & Reference Manual

Document: UM22001
Software Version: 1.0.0
Revision: 0
Date: November 11, 2022

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	ticket_emnet@segger.com*
Internet:	www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: November 11, 2022

Software	Date	By	Description
1.0.0	2022-11-02	YR	Initial release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (compiler, linker, Integrated Development Environment).
- The C programming language.
- The target processor.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table of contents

Chapter 1

Introduction

This chapter provides an introduction to using emVNC. It explains the basic concepts behind emVNC.

1.1 What is emVNC

emVNC is a VNC server optimized to run on embedded devices. It can be used with a graphic library (like emWin) or with a simple pixel storage.

emVNC supports multiple transport layers, TCP/IP or USB can be used to provide a VNC connection to a device.

1.2 emVNC features

emVNC is written in ANSI C and can be used on virtually any CPU. Here is a list of emVNC features:

- Abstract RFB protocol implementation for embedded systems
- Configurable for minimal memory- and flash-footprint
- Usable with custom or SEGGER's off-the-shelf display- and network-modules
- Supports all standard VNC clients, including SEGGER's free, cross-platform emVNC Client
- Supports VNC over USB (e.g. with SEGGER's emUSB-Device stack) or any other socket-like transport layer
- Optional user authentication
- Optional hextile encoding support (less network throughput)
- Optional file transfer support
- Multiple examples for different integrations and use-cases

1.3 Basic concepts

VNC is a remote control tool for GUI environments and graphical applications. Using a VNC connection allows the user to remote control the ’desktop’ of an embedded target device and to interact with it using a PC’s mouse and keyboard. It is based on the RFB protocol as described in RFC6143 and compatible with standard RFB / VNC client implementations.

This server module allows you to connect via any socket-like transport layer (e.g. TCP/IP or USB Bulk) to an embedded device and interact with it via a virtual display. You can use this display like any other display.

You can use a stand-alone virtual display without any real hardware, or you can synchronize it with actually existing display hardware of your embedded device.

1.4 Development environment (compiler)

The CPU used is of no importance; only an ANSI-compliant C compiler complying with at least one of the following international standard is required:

- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

If your compiler has some limitations, let us know and we will inform you if these will be a problem when compiling the software. Any compiler for 16/32/64-bit CPUs or DSPs that we know of can be used. A C++ compiler is not required, but can be used. The application program can therefore also be programmed in C++ if desired.

1.5 Use of undocumented functions

Functions, variables and data-types which are not explained in this manual are considered internal. They are in no way required to use the software. Your application should not use and rely on any of the internal elements, as only the documented API functions are guaranteed to remain unchanged in future versions of the software. If you feel that it is necessary to use undocumented (internal) functions, please get in touch with SEGGER support in order to find a solution.

Chapter 2

Configuring emVNC

This chapter explains how to configure emVNC.

2.1 Using The VNC Server Module

To use this module go through this step-by-step list:

- Include the module in your applications directory structure.
- Let your build-system compile all provided C files when compiling your application.
- Copy and edit the provided `VNC_ConfExample.h` to `VNC_Conf.h` to suit your configuration needs.
- Create an integration layer (see section below)
- Make sure that files in your integration layer can access the VNC.h, `VNC_ConfDefaults.h` and your `VNC_Conf.h` header files.

Integration Layer

The VNC Server Module is a VNC / RFB protocol implementation and is kept as abstract and simple as possible, so that it can be used with any kind of display and over any kind of connection. To use it, you need to fill the following abstractions:

- Configuration
- Session Management (run the server event loop)
- Transport Layer (provide communication with the connected VNC client)
- Display Layer (provide access to the display framebuffer)
- (optional) Event Layer (publishes mouse- and keyboard-events from any connected client)
- (optional) Authentication Layer (generate random data for VNC client challenge-response authentication)

You can find several examples in the examples/ directory, and in the separately provided example pack for the emPower embedded board. These include full integrations for plain framebuffers and for SEGGER's emWin GUI framework, SEGGER's emNet TCP/IP stack, SEGGER's emUSB-Device USB device stack and also for Linux. (The latter mostly for testing of the protocol implementation.)

Detailed information on each of these layers follows. Please refer to VNC.h and `VNC_ConfExample.h` for exact interface definition of the layers that you need to provide.

Configuration

As mentioned, you need to provide a `VNC_Conf.h` that should be based on `VNC_ConfExample.h`. Here you can en/disable optional features, but you also have to provide some type definitions and glue logic for the module.

Session Management

You need to provide a session management that is responsible for accepting/setting up connections when a new client wants to connect, and to run the servers protocol handler function repeatedly for each connection until that client disconnects.

Best practice is to give each client a separate task or thread, such that a blocking network layer doesn't block other tasks. Another option is to only accept a single client connection at a time and to run the main session management and this client connection in a single task.

Overall, for a single client connection you need to setup a `VNC_CONTEXT` with `VNC_ServerInit()`. Then repeatedly run `VNC_HandleClientProtocol()` until a negative value is returned, indicating that the client disconnected or a protocol error forced a disconnect. After that, or at any other time that `VNC_HandleClientProtocol()` is not running, the `VNC_CONTEXT` may be destroyed. Remember to free any other resource related to the client connection, like a socket or a USB bulk handle.

Transport Layer

The transport layer to a connected client is abstracted in `VNC_CLIENT_CALLBACKS` and requires that you provide a callback `VNC_SEND` to send data to the connected client, and `VNC_RECV` to receive data from the client.

Display Layer

The display layer is abstracted in VNC_DISPLAY_CALLBACKS. You need to provide callbacks VNC_GET_SIZE, VNC_GET_PIXEL_FORMAT, VNC_GET_PIXEL and optionally VNC_GET_RECT to return information on the display. You may provide VNC_LOCK and VNC_UNLOCK to allow locking against concurrent access to the display.

Event Layer

The optional event layer allows the server to publish events like key-presses or mouse motion, that a client sends to us. To use it, provide a VNC_EVENT_CALLBACKS with any or all of VNC_MOUSE, VNC_KEYBOARD and VNC_COPY. These will be called when any events are received from the client. To ignore these events, simply set these callbacks to NULL.

Authentication Layer

If password authentication is enabled and a password is set, you need to provide a VNC_AUTH_CALLBACKS structure with VNC_GET_CHALLENGE, which must return 16 random bytes. Otherwise the challenge/response authentication would be predictable.

Note that the authentication mechanism defined in the RFB protocol is by no means secure and should not be relied upon. But using a different, better authentication method would make using standard VNC clients impossible, and increase the server's size and computational needs even further.

Running emVNC

When the target is running the emVNC server and you are using emNet or a different TCP/IP stack for communication any VNC client can be used to connect to your target.

If you are using emUSB for the data transfer SEGGER's emVNC PC client should be used.

Chapter 3

VNC

In this chapter, you will find a description of all API functions as well as all required data and function types.

3.1 Target API

This section describes the functions that can be used by the target application.

Function	Description
General	
VNC_ServerInit()	Initialize VNC_CONTEXT structure so a server-process may be started on it.
VNC_EnableMouseInput()	Enables or disables mouse input via VNC.
VNC_DisplayChanged()	Tells the VNC server that an area of the display was changed.
VNC_EnableKeyboardInput()	Enables or disables keyboard input via VNC.
VNC_HandleClientProtocol()	Check up on client, maintain connection and answer any pending requests.
VNC_RingBell()	Ring a bell on the client if it has one.
VNC_SetLockFrame()	Configures the VNC server not to read the display while the display layer performs drawing operations.
VNC_SetPassword()	Sets the password to use for authentication of clients.
VNC_SetProgName()	Sets the title to be displayed in the title bar of the client window.
VNC_SetRetryCount()	Sets the number of additional trials in case of an error when trying to write data to a client.
VNC_SetSize()	Sets the display size to be transmitted to the client.

3.1.1 VNC_ServerInit()

Description

Initialize VNC_CONTEXT structure so a server-process may be started on it. Call this function to initialize a client connection.

Prototype

```
void VNC_ServerInit(      VNC_CONTEXT           * pContext,
                    const VNC_DISPLAY_CALLBACKS * Display,
                    const VNC_CLIENT_CALLBACKS  * Client,
                    const VNC_AUTH_CALLBACKS    * Auth,
                    const VNC_FILE_CALLBACKS    * File,
                    const VNC_EVENT_CALLBACKS   * Event);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure. This structure is used internally by the VNC server and should not be on the stack.
Display	Pointer to a VNC_DISPLAY_CALLBACKS structure containing display callback function pointers.
Client	Pointer to a VNC_CLIENT_CALLBACKS structure containing client callback function pointers.
Auth	Pointer to a VNC_AUTH_CALLBACKS structure containing authentication callback function pointers. This parameter is only available when VNC_SUPPORT_AUTH is enabled.
File	Pointer to a VNC_FILE_CALLBACKS structure containing file callback function pointers. This parameter is only available when VNC_SUPPORT_FILE is enabled.
Event	Pointer to a VNC_EVENT_CALLBACKS structure containing display callback function pointers.

3.1.2 VNC_EnableMouseInput()

Description

Enables or disables mouse input via VNC.

Prototype

```
void VNC_EnableMouseInput(VNC_CONTEXT * pContext,
                          int           OnOff);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
OnOff	1 for enabling mouse input, 0 for disabling.

3.1.3 VNC_DisplayChanged()

Description

Tells the VNC server that an area of the display was changed.

Prototype

```
void VNC_DisplayChanged(VNC_CONTEXT * pContext,
                        unsigned      x,
                        unsigned      y,
                        unsigned      Width,
                        unsigned      Height);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
x	Base coordinate of changed area
y	Base coordinate of changed area
Width	Width of changed area
Height	Height of changed area

Notes

MUST NOT be called while holding the display lock.

3.1.4 VNC_EnableKeyboardInput()

Description

Enables or disables keyboard input via VNC.

Prototype

```
void VNC_EnableKeyboardInput(VNC_CONTEXT * pContext,
                             int           OnOff);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
OnOff	1 for enabling keyboard input, 0 for disabling.

3.1.5 VNC_HandleClientProtocol()

Description

Check up on client, maintain connection and answer any pending requests. Call this function regularly for every client to maintain the client connection.

Prototype

```
int VNC_HandleClientProtocol(VNC_CONTEXT * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.

Return value

> 0	OK, a delay of this milliseconds is suggested before recalling the event handler.
= 0	OK
< 0	error

Notes

pContext->aOutBuffer MUST be empty when calling. It will be used, but is guaranteed to be empty afterwards again.

3.1.6 VNC_RingBell()

Description

Ring a bell on the client if it has one.

Prototype

```
void VNC_RingBell(VNC_CONTEXT * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.

3.1.7 VNC_SetLockFrame()

Description

Configures the VNC server not to read the display while the display layer performs drawing operations.

Prototype

```
void VNC_SetLockFrame(VNC_CONTEXT * pContext,
                      unsigned      OnOff);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
OnOff	If set to a value >0 frame locking will be enabled.

Additional information

This can be configured at compile time by using the compile time switch VNC_LOCK_FRAME.

Notes

The default of this is set via VNC_LOCK_FRAME.

3.1.8 VNC_SetPassword()

Description

Sets the password to use for authentication of clients

Prototype

```
void VNC_SetPassword(      VNC_CONTEXT * pContext,
                     const char        * sPassword);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
sPassword	Location of password to use. Set to NULL to disable authentication.

Notes

sPassword must remain valid until server is terminated or a new password is set. It is NOT copied to a local buffer.

3.1.9 VNC_SetProgName()

Description

Sets the title to be displayed in the title bar of the client window.

Prototype

```
void VNC_SetProgName(      VNC_CONTEXT * pContext,
                     const char        * sProgName);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
sProgName	Title to be displayed in the title bar of the client window.

3.1.10 VNC_SetRetryCount()

Description

Sets the number of additional trials in case of an error when trying to write data to a client.

Prototype

```
void VNC_SetRetryCount(VNC_CONTEXT * pContext,
                       unsigned      Count);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
Count	Number of additional trials to be used in case of an error (default is 0).

3.1.11 VNC_SetSize()

Description

Sets the display size to be transmitted to the client.

Prototype

```
void VNC_SetSize(VNC_CONTEXT * pContext,
                 unsigned      Width,
                 unsigned      Height);
```

Parameters

Parameter	Description
pContext	Pointer to a VNC_CONTEXT structure.
Width	X-size to be used.
Height	Y-size to be used.

Additional information

This function must be called between VNC_ServerInit() and VNC_ServerRun(). The size passed to this function may be smaller than the real display. If the access methods (pfGet-Pixel) supports out-of-bounds access, the size may also be larger than the actual display. Per default the server uses the layer size.

3.2 Structures

The table below lists the available structures.

Structure	Description
VNC_DISPLAY_CALLBACKS	Callbacks used to handle display operations.
VNC_CLIENT_CALLBACKS	Callbacks used for data transfer.
VNC_EVENT_CALLBACKS	Callbacks used to handle VNC events.
VNC_AUTH_CALLBACKS	Callbacks used for authentication.

3.2.1 VNC_DISPLAY_CALLBACKS

Description

Callbacks used to handle display operations.

Type definition

```
typedef struct {
  VNC_LOCK             * pfLock;
  VNC_UNLOCK           * pfUnlock;
  VNC_GET_SIZE         * pfGetSize;
  VNC_GET_PIXEL_FORMAT * pfGetPixelFormat;
  VNC_GET_PIXEL        * pfGetPixel;
  VNC_GET_RECT         * pfReadRect;
} VNC_DISPLAY_CALLBACKS;
```

Structure members

Member	Description
pfLock	Lock display.
pfUnlock	Unlock display.
pfGetSize	Get display size.
pfGetPixelFormat	Get pixel format.
pfGetPixel	Get a single pixel.
pfReadRect	[Optional] Get rectangle.

3.2.2 VNC_CLIENT_CALLBACKS

Description

Callbacks used for data transfer.

Type definition

```
typedef struct {
  VNC_SEND * pfSend;
  VNC_RECV * pfRecv;
} VNC_CLIENT_CALLBACKS;
```

Structure members

Member	Description
pfSend	Send data to client.
pfRecv	Receive data from client.

3.2.3 VNC_EVENT_CALLBACKS

Description

Callbacks used to handle VNC events.

Type definition

```
typedef struct {
  VNC_MOUSE    * pfMouse;
  VNC_KEYBOARD * pfKeyboard;
  VNC_COPY     * pfCopy;
} VNC_EVENT_CALLBACKS;
```

Structure members

Member	Description
pfMouse	[Optional] Mouse callback.
pfKeyboard	[Optional] Keyboard callback.
pfCopy	[Optional] Copy callback.

3.2.4 VNC_AUTH_CALLBACKS

Description

Callbacks used for authentication.

Type definition

```
typedef struct {
  VNC_GET_CHALLENGE * pfGetChallenge;
} VNC_AUTH_CALLBACKS;
```

Structure members

Member	Description
pfGetChallenge	[Optional] Handle authentication.

3.3 Function Types

The table below lists the available function types.

Type	Description
VNC_SEND	Send data to client.
VNC_RECV	Receive data from client.
VNC_LOCK	Protect display against concurrent accesses.
VNC_UNLOCK	Remove lock placed by VNC_LOCK.
VNC_GET_SIZE	Retrieves the display size.
VNC_GET_PIXEL_FORMAT	Retrieves the pixel format.
VNC_GET_PIXEL	Get color value for a specified pixel.
VNC_GET_RECT	Optional callback to retrieve multiple pixels in one go.
VNC_MOUSE	Handle mouse event coming from client.
VNC_KEYBOARD	Handle keyboard event coming from client.
VNC_COPY	Handle client copied text.
VNC_GET_CHALLENGE	Handle authentication.

3.3.1 VNC_SEND

Description

Send data to client.

Type definition

```
typedef int (VNC_SEND)(      VNC_CONTEXT * pContext,
                       const U8          * pBuffer,
                             unsigned      Len);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
pBuffer	Pointer to the data to be sent.
Len	Number of bytes to be sent.

Return value

> 0	Amount of data sent.
= 0	Disconnected from the client.
< 0	An error occurred.

3.3.2 VNC_RECV

Description

Receive data from client.

Type definition

```
typedef int (VNC_RECV)(VNC_CONTEXT * pContext,
                       U8          * pBuffer,
                       unsigned      Len);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
pBuffer	Pointer to a buffer.
Len	Number of bytes to read.

Return value

> 0 Amount of data received.
= 0 Disconnected from the client.
< 0 An error occurred.

3.3.3 VNC_LOCK

Description

Protect display against concurrent accesses.

Type definition

```
typedef void (VNC_LOCK)(VNC_CONTEXT * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.

3.3.4 VNC_UNLOCK

Description

Remove lock placed by VNC_LOCK.

Type definition

```
typedef void (VNC_UNLOCK)(VNC_CONTEXT * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.

3.3.5 VNC_GET_SIZE

Description

Retrieves the display size.

Type definition

```
typedef void (VNC_GET_SIZE)(VNC_CONTEXT * pContext,
                            unsigned    * pWidth,
                            unsigned    * pHeight);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
pWidth	in Display width.
pHeight	in Display height.

3.3.6 VNC_GET_PIXEL_FORMAT

Description

Retrieves the pixel format. The callback must store the correct format inside VNC_CONTEXT->PixelFormat.

Type definition

```
typedef int (VNC_GET_PIXEL_FORMAT)(VNC_CONTEXT * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.

Return value

= 0 Success.
≠ 0 Error.

3.3.7 VNC_GET_PIXEL

Description

Get color value for a specified pixel.

Type definition

```
typedef U32 (VNC_GET_PIXEL)(VNC_CONTEXT * pContext,
                            unsigned      x,
                            unsigned      y);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
x	Pixel coordinate.
y	Pixel coordinate.

Return value

U32 color value, in device format.

Additional information

The function must perform a color conversion if necessary.

3.3.8 VNC_GET_RECT

Description

Optional callback to retrieve multiple pixels in one go.

Type definition

```
typedef void (VNC_GET_RECT)(VNC_CONTEXT * pContext,
                            unsigned      x,
                            unsigned      y,
                            unsigned      xSize,
                            unsigned      ySize,
                            U32         * pPixelBuffer);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
x	Pixel coordinate.
y	Pixel coordinate.
xSize	Rectangle x size
ySize	Rectangle y size
pPixelBuffer	Pointer to a buffer to store the pixel data (pixel index only, no color conversion)

Additional information

The function should not perform a color conversion. When using this function the color format of the client and the color format used for the display must match.

3.3.9 VNC_MOUSE

Description

Handle mouse event coming from client.

Type definition

```
typedef void (VNC_MOUSE)(VNC_CONTEXT * pContext,
                         unsigned      x,
                         unsigned      y,
                         unsigned      PressedButtonMap);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
x	Mouse coordinate.
y	Mouse coordinate.
PressedButtonMap	Map of buttons currently pressed on the mouse.

3.3.10 VNC_KEYBOARD

Description

Handle keyboard event coming from client.

Type definition

```
typedef void (VNC_KEYBOARD)(VNC_CONTEXT * pContext,
                            U16           Key,
                            char          IsPressed);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
Key	VNC key code.
IsPressed	1 - key is pressed, 0 - key is released.

3.3.11 VNC_COPY

Description

Handle client copied text.

Type definition

```
typedef void (VNC_COPY)(VNC_CONTEXT * pContext,
                        char        * pBuffer,
                        unsigned      Length);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
pBuffer	Pointer to a buffer containing the copied text.
Length	Length of the data in the buffer.

3.3.12 VNC_GET_CHALLENGE

Description

Handle authentication.

Type definition

```
typedef void (VNC_GET_CHALLENGE)(VNC_CONTEXT * pContext,
                                 U8          * pChallenge);
```

Parameters

Parameter	Description
pContext	Pointer to a valid VNC_CONTEXT structure.
pChallenge	Pointer to a buffer containing the authentication challenge.

Additional information

This must save a challenge of VNC_AUTH_CHALLENGE_SIZE bytes in pChallenge. Usually that should be newly generated random data.

Chapter 4

Support

4.1 Contacting support

Before contacting support please make sure that you are using the latest version of the emVNC package. Also please check the chapter *Configuring debugging output* on page and run your application with enabled debug support.

If you are a registered emVNC user and you need to contact the emVNC support please send the following information via email to ticket_emnet@segger.com*:

- Your emVNC registration number.
- emVNC version.
- A detailed description of the problem
- The configuration files
- Any error messages.

Please also take a few moments to help us to improve our service by providing a short feedback when your support case has been solved.

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

