VNC

Thomas the Richter

COLLABORATORS

	TITLE : VNC		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas the Richter	April 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

VNC

1.1 ViNCEd Guide

___ ___ ___ _____ ________ _ | | |\ | / \ | | | | | * | \ | / -- | | | | | \ | | | _____| | | | | \ | | |____ / | \ / | | \ | | | | | \ / | | \ | | | | | \ / | | \ | \ -- | | | \/ __|__ _|_ \| _____/ |________| ____/|

Version 3.67 ____

ViNCEd - Guide 4.15 ViNCEd 3.67

© 1990-1998 THOR-Software Thomas Richter Rühmkorffstraße 10 A 12209 Berlin

Germany

EMAIL: thor@einstein.math.tu-berlin.de (You tried the FAQ, did you?)

WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

1.2 The THOR-Software Licence

1.3 The User's Guide

1.4 The SetVNC Tool

REMARK: The SetVNC program uses overlays to minimize memory usage and disk loading time. Please DO NOT make resident and DO NOT crunch! Some virus checkers have problems testing overlayed programs. THIS IS NOT A BUG in SetVNC, but a bug in the virus checkers!

1.5 The Programmer's Manual

The Programmer's Manual: The prime source for the real freak

1.6 My address

Thomas Richter Rühmkorffstraße 10 A 12209 Berlin

Germany

EMAIL: thor@math.tu-berlin.de WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

When sending EMail, make sure that the reply address is valid and your name and address contains only ASCII (7-Bit) characters. Since we receive by far too much "junk mail", EMail with invalid header or invalid reply address is bounced automatically. Especially, due to lots of spam received thru "hotmail", this provider has been "banned" by our mail server.

1.7 Book references

For a (not so) complete documentation of all CON: control sequences, read

The Amiga ROM Kernal Reference Manual, Volume Devices. 3rd Edition. Addison-Wesely Publishing Company, Inc. ISBN 0-201-56755-X

For a list of recommended user interfacing methods, read

The Amiga ROM Kernal Reference Manual, Volume User Style Interface Style Guide. 1st Edition. Addison-Wesely Publishing Company, Inc. ISBN 0-201-57757-7

For a description of the Amiga hardware in general and the keyboard raw key codes in special, read

The Amiga Hardware Reference Manual. 3rd Edition. Addison-Wesely Publishing Company, Inc. ISBN 0-201-56776-8

For a (poor) documentation of AmigaDOS and packet types:

The AmigaDOS Manual, 3rd Edition. Bantam Books (The Bantam Amiga Library) ISBN 0-553-35403-4

For an (excellent) documentation of AmigaDOS and packet types:

The Amiga Guru Book Ralph Babel, Taunusstein

(however, handle this book with some care....)

Additional information about the XTerm and WinTerm control sequences was taken from the manual pages and the documentation of XTerm and WinTerm.

The XTerm docs can be found on every well installed unix system, since this program is a standard tool. The WinTerm docs can be found on SGI systems only.

However, the location of these docs may vary from system to system. You usually obtain them by "man xterm" or "man winterm" - if the documentation is installed.

1.8 Overview: What is ViNCEd about? And why this strange name?

ViNCEd is a console handler replacement for the console device and the related DOS drivers, CON: and RAW:. So much nothing new, you say?

There is a difference, a big difference. You'll notice that as soon as you work with ViNCEd. It's different, really different - ViNCEd is a full screen console handler with many additional features.

As for this full screen thing: The whole window contents is kept in memory, so you may use the cursor keys to move upwards on the screen and may re-execute commands you typed some time ago by pressing the return key again in the input line. Reminds you on some 8 bit computers? You're right! But there's more...

You can cut, copy and paste text to and from the clipboard, as with the standard handler. You may insert text quickly with the middle mouse button or a replacement sequence, you may ask ViNCEd, too, to copy the text automatically as soon as you marked it. If you're used to the Unix XTerm program, you'll appreciate this feature. But there's more...

ViNCEd comes with two scrollers, to scroll the window contents left and right, up and down. ViNCEd keeps a limited, but configurable amount of the input and output for you. Available for re-execution by the return key. The buffer contents can be saved and reloaded if required, the buffer size can be configured. There's of course a command history as usual. But there's more...

ViNCEd offers "Macros" and "Buttons". These are short keyboard sequences that can be invoked by either a gadget in the window title, a keystroke or a menu item. Macros can not only replace keyboard sequences - they can also show requesters and expand wild cards for you. ViNCEd uses also build-in macros for all-day purposes like closing the shell, or calling a configuration program. But there's more...

The Shell mode of ViNCEd comes with up to six configurable TAB expansion keys, each of them can be setup individually with an optional file requester, individual priorities for the files to be searched, and a TAB expansion cache for quick disk access. But there's more...

ViNCEd offers a Ctrl-Z key and job control for the standard Amiga shell. That means, in case you wonder, that you can always invoke a new shell in a ViNCEd window, even if another program blocks the current shell. You can switch back and forth between the different shells in the programs with two unix like commands "fg" and "bg". But there's more...

ViNCEd comes with a fully configurable keyboard. If you don't like the control keys where they are, just use a different set. Configuration of the keyboard can be done thru a graphical user interface, the SetVNC program. But there's more...

ViNCEd understands far more control codes than the ordinary console. It offers a VT-220 compatibility mode which might come handy for terminal programs. A lot of private commands and XTerm commands round up this set. And there is still more...

ViNCEd supports graphics cards and customizable color sets. ViNCEd windows are, unlike the console windows, not restricted to the first eight pens. It comes with a set of sixteen configurable pens, which can be setup in a way that they match the ANSI standard. ViNCEd can be made to appear on its own screen, which can be made public, if required. The screen mode for this screen is under your control, of course. The window font is too. And there is still more...

Ooops, just forgot the most important point... It's free. Well, at least for the private user - please check the licence agreement .

Oh yes, about this strange name, I forgot to tell you: The first version of ViNCEd appeared years ago (even before 1990, so this is a very old program, indeed). In lack of a better name, I called it VeryNewCon - as opposed to the NEWCON: developed by Commodore for the 1.3 Workbench which was the latest clue at this time. VeryNewCon was short VNC: - but since this name is really boring, I decided to fill in some vowels - and add an "Ed". Well, it's an editor which runs, apparently, a shell in it. Not just a console...

1.9 How to install ViNCEd

Installation is easiest with the supplied "Installer" script in the archive. In expert mode, it will ask you about each step and won't hurt your system at all. However, in case you have to install by hand, for whatever reason, look at the "InstallVNC" file in the "S" drawer of the distribution. The following steps are required:

o) open a shell, create a new directory and extract the archive into that directory.

o) copy the "vnc.library" from the "libs" drawer of the distribution to a location where libraries will be found, i.e. "LIBS:".

o) copy the "SetVNC" program from the "c" drawer of the distribution to a place where it will be found by shell, i.e. somewhere in the command path.

o) copy the "ViNCEd.guide" to a place where you want to keep it.

o) copy the "SetVNC.info" icon from the "prefs" drawer of the distribution to a place where you keep your preferences. Edit this icon by the workbench. Adjust the "Default Tool" to the location where you put the SetVNC program. Adjust the "Helppath" tooltype to the location where you stored the guide.

o) copy part of the localizations in the "Locale/Catalogs" drawer to the LOCALE: drawer of your workbench. No other place will work!

o) copy the files in the "Devs/Dosdrivers" directory of the distribution to a place where mount icons will be found. This is usually "DEVS:Dosdrivers" or "Storage/Dosdrivers". "VNC" and "NEWCON" mount ViNCEd as a CON: replacement under the names "VNC" resp. "NEWCON", "VNR" is a "raw version" of ViNCEd. If you're using Workbench 2.0, edit the "Devs:MountList" file and insert the files "DEVS/Mount_VNC" and/or "DEVS/Mount_NEWCON" in the distribution.

o) copy the scripts "fg", "bg", "fork", "history", "More", "SetKeyboard" and "SetFont" in the "S" drawer of this distribution to a place where shell scripts will be found. Set the "s" bit by hand if it possibly got lost by "lha". Change your path in a way such that these scripts are found BEFORE the CBM "More", "SetFont" and "SetKeyboard" commands. These scripts replace ugly and hackish implementations of the CON: related commands.

o) if you want to replace the standard console handler CON: completely by ViNCEd, add the following line to your startup-sequence or user-startup:

SetVNC quiet mount override as CON:

You may also remove the "ConClip" command in this case.

o) if you want to replace the "Shell" icon, install the ViNCEd shell icon in the distribution. If not, and you don't want to replace CON: with ViNCEd completely, edit the "Shell" icon of your system. Adjust or add the following tooltype:

WINDOW=VNC:0/0/-1/-1/AmigaShell/SHELL/CLOSE

o) if you want to keep the vnc.library documentations, read the ReadMe in the "Include" drawer and install it where you want to keep it.

o) if you're still working with Os 2.0 or Os 2.1 and want to use the Ctrl-Z function of ViNCEd, copy the NamedConsoleHandler from the "Extras" directory into the L: directory of your boot partition, and copy the "CONSOLE" icon and mount entry from the same drawer to DEVS:DosDrivers. DO NOT CHANGE ITS NAME. Make sure this "CONSOLE" is mounted on startup.

o) if you want to patch "More" to allow iconification of windows "More" run in, you should enter the following command:

Extras/SPatch -ot:More -pExtras/More.pch SYS:Utilities/More

Then copy t:More on top of the original "More" program. This requires, however, the original and unmodified 40.3 (Os 3.1) version of "More".

o) correct a bug in the rexxsyslib.library: Copy the original rexxsyslib.library to a safe place, then go into the directory where you extracted the archive. Enter the following command:

Extras/SPatch -ot:rexxsyslib.library -pARexx/rexxsyslib.pch LIBS:rexxsyslib.library

Then copy the t:rexxsyslib.library on top of the LIBS:rexxsyslib.library. This fixes an ARexx bug (which is however not related to ViNCEd).

o) if you want to install the "StringSnip" program: This program improves the behavour of "String gadgets" and makes most ViNCEd style keyboard commands available in string gadgets, too. Copy the file "Extras/StringSnip" file to a directory where commands will be found. Then add the following command to your startup-sequence:

StringSnip >NIL: install

o) if you want to use Massimo's ToolButtonClass - ViNCEd uses this to get an image for its iconification gadget - copy this from the "Extras" drawer to "SYS:Classes/Images"

o) if you want to install the "UnixDirs3" patch by "Timo Kaikumaa" (thanks, Timo, by letting me include this program here!): Read the "UnixDirs3" guide in the Extras/UnixDirs3 drawer first. This program modifies certain AmigaDos functions in a way that make "unix" path conventions available to AmigaDos. Installation is simple, just drag the "UnixDirs3" icon in the "Extras/UnixDirs3" directory into your "WBStartup" drawer. Copy the "UnixDirs3" guide to a place where you want to keep it. Disable other patches that try to improve the AmigaOs in a similar way. I checked most of them and most of them are buggy in one or another way. "UnixDirs3" has proven to work reliable here. Especially, if you see "SetVNC" crash on loading, it might be the fault of a broken patch!

o) you are HIGHLY ADVICED to install the "TrueMultiAssigns" program as well. It fixes severals flaws of the AmigaOs concerning multi-directory-assigns. Without it, TAB expansion in multi assigns might behave a bit strange. If you're running other patches that try to fix this as well, DISABLE THEM. I tried various, but none of them worked as reliable as they should. This patch has been successfully tested with ViNCEd and the rest of the system.

For installation, copy the file "Extras/TrueMultiAssigns" to a location where it is found by the shell, i.e. into the command path. Then add the following line to your startup-sequence:

TrueMultiAssigns

o) if you want to keep it: Install the "topaz6.font" of this distribution. It replaces the "xen font" which is, unlike what you might think, a proportional font and does therefore not work with ViNCEd. To do so, go into the directory where you've extracted this archive to and enter the following command:

copy Fonts/topaz6#? to FONTS: all

o) reboot your system. That's it.

1.10 The Keyboard - how to type?

How to type then? Sounds like a silly question, I guess... However, there's somewhat more to be said than pressing the "a" key prints an "a" on the screen, since the keyboard in a ViNCEd window is configurable. This means a lot more than just selecting a national keyboard as with the Preferences "Locale" editor. It means that you can

Bind Keyboard Functions to Keyboard Keys.

The reason why the "a" key inserts an "a" is now that all standard ViNCEd settings DO NOT bind a function to the "a" key, which means that the "a" key invokes the default function it was given by the Os - namely to print an "a". As for ViNCEd, you can easely bind this key to the Cursor Up function . If you really want to....

Here's, for first, the functions the keyboard keys invoke as long as they are NOT BOUND to any function explicitly. You will rarely see these functions because even the "Default Keyboard Settings" which are fixed down below in ViNCEd bind at least some of the keys. However, since this "binding" is done at ViNCEd level, you can change these "default bindings".

The Default Key Binding is Something Different than Unbound Keys.

Here now the list of functions unbound keys invoke: (see also: Keyboard notations)

All white keys on the main keyboard, characters, numbers and the keys on the numeric keypad:

The character that was bound to this key by the keymap of the window. This is usually just the letter that is printed on the key.

The cursor keys:

Moves the cursor in the window in the direction of the arrow.

The function keys:

Invokes the button macros in the title screen.

The Tab key:

Inserts or jumps to the next tabulator stop.

The Shift Tab combination:

Deletes characters up to or jumps to the previous tabulator stop.

The Help key:

Launches the help macro .

The Return key:

Collects the inputs of the line the cursor is placed in and sends them as inputs to whichever program listens; inserts a new line below the current. This is just the standard function you expect from the console.

The Backspace key:

This erases the next character to the left of the cursor, if this character has been typed by you. It will stop erasing printed characters.

The Del key:

This key erases the character under the cursor and scrolls the characters to the right inwards. It won't delete printed characters but will only remove input characters.

The combination Shift cursor:

Moves the cursor half the window to the left, right, up or down.

The combination Ctrl a:

Moves the cursor to the start of the user inputs, to the left edge.

The combination Ctrl c:

Sends the break signal (signal 12) to the current process and the background processes. This should abort a running program.

The combination Ctrl d:

Sends the stop signal (signal 13) to the current process and the background processes. This should abort a running shell script.

The combination Ctrl e:

Sends the signal 14 to the current process and the background processes.

The combination Ctrl f:

Sends the signal 15 to the current process and the background processes.

The combination Ctrl h:

Identically to the Backspace key.

The combination Ctrl i:

Identically to the Tab key.

The combination Ctrl j:

Insert a new line under the current line, but does not sent any inputs.

The combination Ctrl k:

Cuts the user inputs starting at the cursor position, and places them in the Yank buffer .

The combination Ctrl l:

Erases the contents of the visible window and the lines below the visible window. This is called the "Line Feed" window.

The combination Ctrl q:

Resumes a with Ctrl s stopped output. This is also known as the "XON" function.

The combination Ctrl m:

Identically to the Return key.

The combination Ctrl r:

Searches the text in front of the cursor position in the command history .

The combination Ctrl s:

Stops the output. This is known as the "XOFF" function.

The combination Ctrl u:

Cuts all user inputs to the left of the cursor position and inserts them into the Yank buffer .

The combination Ctrl w:

Erases the word to the left of the cursor and places it into the Yank buffer .

The combination Ctrl x:

Cuts the complete user inputs of the current line, and inserts them into the Yank buffer .

The combination Ctrl y:

Inserts the contents of the Yank buffer .

The combination Ctrl z:

Forks the a new shell in the current window. More about the Ctrl-Z function is found at a different location .

The combination Ctrl \:

Invokes either a macro or sends an "End Of File" to the current shell or the program running in the window. The details depend on a preferences flag .

The next keys are not available on a standard amiga keyboard. However, a replacement keyboard with a properly setup system keymap definition table may offer these keys. ViNCEd is at least able to handle them correctly. Even though these keys might be not available for you, you can still bind other keys to the keyboard functions they generate.

Insert:

Toggles between overwrite and insertion mode.

Shift Insert:

Quite the same; toggles between overwriting and insertion.

Page Up:

Moves the cursor upwards half a window.

Shift Page Up:

Scroll the window upwards half its size. Tries not to move the cursor if this is possible.

Page Down:

Moves the cursor downwards half a window.

Shift Page Down:

Scrolls the screen downwards half a window.

Pause/Break:

Pauses the output. If this key is pressed again, the output is resumed.

Shift Pause/Break:

Always re-enables output, similar to Ctrl q.

Home:

Moves the cursor to the beginning of the buffer, to the top, left edge.

End:

Moves the cursor to the end of the buffer, to the right, bottommost position.

Again, as mentioned above, this is just the list of the keyboard functions for "unbound" keys. Even the ViNCEd "Default" configuration binds some keys for your convenience. However, this keyboard configuration is completely up to you and can be modified with the SetVNC tool.

Here's now the list of keyboad bindings in the default configuration. The meaning of the keyboard functions, if not obvious, can be found elsewhere or by following the links. The qualifier "NumL" means that this keyboard function is only available if NumLock is active. This does NOT necessarily relate to the numeric keypad, even though in the default configuration only keys of the numeric keypad make use of this qualifier. Keys on the numeric keypad itself are identified by a "Num" in front of them, as for example Num3, which is the key "3" on this pad. The sequence NumL Num3 means hence that the key "3" on the numeric pad is bound to the following function ONLY IF NumLock is active.

Cursor left: Cursor Left

Cursor right: Cursor Right

Cursor up: Cursor Up

Cursor down: Cursor Down

The binding of these keys could have been avoided because the unbound keys provide the same function.

Alt Cursor Up History Up

Alt Cursor Down: History Down

Ctrl r: Search Partial Upwards

Shift Alt Cursor up: Search History Upwards

Shift Alt Cursor down: Search History Downwards

The next four keyboard bindings are again superfluous:

Shift Cursor left: Half Screen Left

Shift Cursor right: Half Screen Right

Shift Cursor up: Half Screen Up

Shift Cursor down: Half Screen Down

Ctrl Cursor left: To Left Border

Ctrl Cursor right: To Right Border

Ctrl Cursor up: To Top of Screen

Ctrl Cursor down: To Bottom of Screen

Ctrl Cursor left: Prev Word

Ctrl Cursor right: Next Word

Ctrl Alt Cursor left: Prev Component

Ctrl Alt Cursor down: Next Component

Ctrl Alt Cursor up: Scroll Up

Ctrl Alt Cursor down: Scroll Down

Ctrl Shift Alt Cursor up: Scroll Half Screen Up

Ctrl Shift Alt Cursor Down: Scroll Half Screen Down

NumL Num4: Cursor Left

NumL Num6: Cursor Right

NumL Num8: Cursor Up

NumL Num2: Cursor Down

NumL Num9: Half Screen Up

NumL Num3: Half Screen Down

NumL Home3: Home

NumL End: End

Return: Send Inputs

Alt Return: Split Line

Shift Alt Return: Insert ˆJ

Ctrl Return: Send Complete Line

Shift Return: Line Feed

The next two key bindings are again superfluous because the unbound keys provide the same function:

Tab: TAB Forwards

Shift Tab: TAB Backwards

Ctrl Tab: Expand Path

Ctrl Shift Tab: Expand Backwards

Ctrl LAlt Tab: Expand Short

Ctrl RAlt Tab: Expand Devices

Ctrl Shift RAlt Tab: Expand Devs Bkwds

Ctrl Shift C: Send ˆC

Ctrl Shift D: Send ˆD

Ctrl Shift E: Send ˆE

Ctrl Shift F: Send ˆF

The next six keys bindings are again unnecessary. If you need room in your keyboard definition, you may for example leave the following four keys undefined without loss of any function.

Ctrl c: Send ˆC to All

Ctrl d: Send ˆD to All

Ctrl e: Send ˆE to All

Ctrl f: Send ˆF to All

Del: Delete Forwards

Backspace: Delete Backwards

Shift Del: Delete Full Line

Alt Del: Delete Inputs

Alt Backspace: Delete Word Bkwds

Ctrl Alt Del: Delete Component Fwds

Ctrl Alt Backspace: Delete Component Bkwds

Shift Alt Del: Delete End of Line

Shift Alt Backspace: Delete Start of Line

The next binding is again not required because the function bound to the key is identical to the function of the unbound key:

Ctrl l: Form Feed

The next nine functions are available in the menu anyways. However, they are bind here because they should work even if no menu is attached to a ViNCEd window.

RAmiga e: Delete End of Display

RAmiga l: Clear Screen

RAmiga x: Cut

RAmiga c: Copy

RAmiga v: Paste

RAmiga h: Hide

RAmiga a: Select All

RAmiga r Reset

RAmiga Shift R Full Reset

The next seven bound function are again identical to the function of the unbound keys:

Esc: Toggle ESC

Ctrl s: Suspend

Ctrl q: Resume

Ctrl y: Yank

Ctrl \: Generate EOF

Ctrl z: Fork New Shell

Help: Help

Alt Num[: Toggle NumLock

Alt Num0: Toggle Overwrite

Alt Esc: Insert CSI

Shift Esc: Insert ESC

This is just the default keyboard configuration, there are lots of other keyboard functions available than listed here, check for details the documentation of the keyboard pages of the SetVNC program.

1.11 Non standard keys

This key is usually not found on the Amiga keypad. However, a non-standard keymap may create the standard ANSI control sequence that is understood by ViNCEd to invoke the mentioned function.

1.12 The Yank buffer

The "Yank Buffer" is a one line internal ViNCEd character buffer. It is set by various "Cut" commands and can be inserted any time by the Yank function . The default configuration binds this function to the Ctrl Y keyboard combination.

The contents of this buffer is never placed in the clipboard, it is kept completely ViNCEd internal.

1.13 The notation of keyboard sequences in this manual

All words printed in reverse colors represent single keyboard keys or keyboard combinations. A single key is just represented by a single character, or a single word - the same text that is printed on this key on your keypad. As for example:

Ctrl:

The "Control" key.

a:

The "a" key.

Keyboard combinations consist of a list of "modifier keys", such as Shift, Alt, Amiga or Ctrl, and an ordinary key. All these keys have to be pressed AT ONCE. For example:

Ctrl y:

The "Control" key, pressed together with the "y" key.

And some trickier combinations:

Ctrl Shift Alt Tab:

The "Control", "Shift", "Alternate" and "Tab" key, all to be pressed at once.

A somewhat special qualifier is NumL. Unlike what you might think, this DOES NOT relate to keys on the numeric keypad. It says that this function is available as long as "NumLock" is active. Unfortunately, there's no light on the keyboard available to make the state of this qualifier visible, but ViNCEd keeps it nevertheless. The keyboard function Toggle NumLock is available to change the state of the NumL qualifier.

The keys on the numeric keypad, however, are identified by a "Num" in front of their name, as for example Num5 for the digit "5" in the middle of the pad, or Num[for the square bracket in the left top corner.

All the other qualifiers are easier to understand:

Shift: Either the left or the right shift key.

LShift: Only the left shift key.

RShift: Only the right shift key.

Alt: Either the left or the right alternate key.

LAlt: Only the left alternate key.

RAlt: Only the right alternate key.

LAmiga: The left amiga key. This is the Commodore key on some keyboards.

RAmiga: The right amiga key.

Ctrl: The control key.

1.14 What is a word?

A word is just any sequence of characters that is separated by blank spaces. No quotation marks or other special characters are parsed. That is, one shell argument may consist of several words in this sense.

1.15 What is a component?

A component is any sequence of characters that is separated by either blank spaces, forward slashes "/" or colons ":". Thus, a component is almost the same as a path component of an Amiga Dos file name except for the case the file name contains blanks.

Please note that all component relevant keyboard functions are only available in Shell mode . If the Shell mode is disabled, a component is identically to a word .

1.16 The Break functions of ViNCEd

There's some difference in philosophy which programs should be "broken" and which not. ViNCEd supports both philosophies, the "Amiga standard" as well as the "break force" method.

The "Amiga standard" method is to sent the break signals only to "foreground processes", i.e. programs that have been started directly in the shell. Programs that have been launched with the "Run" command will not receive these signals and can't be aborted for this reason by a standard Amiga shell. I regard this as a bit dangerous because I won't be able to stop operations in emergency operations. Therefore, the default keyboard configuration, as well as the unbound break keys will use the ViNCEd "break force" method which sends the signal bits to both, the foreground and the background processes.

However, you don't need to join my opinion and can bind the break keys to the "standard Amiga" functions which operate a bit more restrictive.

1.17 What is a keyboard function?

A keyboard function is an "action" ViNCEd can perform in its window which is invoked whenever a certain key is pressed. The simplest example of a keyboard function is the Cursor Right function. Whenever it is invoked, the cursor is moved one position to the right.

However, there are fairly more complicated functions than this one: Take the Tab Expansion as another example. It does quite a lot more than just moving the cursor - it searches directory trees, sorts the found file names and inserts an expansion back into the window.

Keyboard functions can be bound to keys of the keyboard - that's what they are used for. However, even the keys that are not bound to any function at all will perform something, of course, sometimes even functions you may bind to other keys explicitly. You won't be able to type anything in the window if they won't - the standard alpha- numerical keyboard keys perform simply text insertion.

There is also a list of available keyboard functions in this guide.

1.18 Binding a keyboard function?

"Binding" means to tell ViNCEd that a certain key should invoke a certain keyboard function . This "binding" is done by the preference editor that comes with ViNCEd, namely SetVNC , or the Keyboard pages of this program, to be precise.

All the details how to setup a custom keyboard can be found there, more about the keyboard in general is in the keyboard section .

1.19 Gadgets in the Window

A standard ViNCEd window comes with five sets of "gadgets", i.e. clickable areas in the window frame. For the first, there are the system standard gadgets used to move and place the window, to resize it, to "depth arrange" it and to close it. I guess I need not to say how they work, but I should say that you can specify which system gadgets should be added when a ViNCEd window is opened. This is done thru options placed in the window path and is described in detail elsewhere. Even defaults can be setup thru the SetVNC preference editor.

A non-standard gadget is the iconify gadget in the top edge of the window, to the left of the "depth arrange" gadgets. If pressed, it will turn the window into an "icon" that is placed on the workbench - which is sometimes useful if you want ViNCEd to get "out of the way". All output can continue, even if the window is "iconified". This gadget can be turned on or off by default as well, again by SetVNC .

However, there's something that needs to be said about iconification: Certain programs put the window in a state where it cannot get iconified anymore. There is not much I can do about it except providing "workarounds". The most common problem is the program "more"; a replacement script that fixes this feature has been put into your S: directory as part of the installation procedure, and this script should be used as replacement. For details, consider reading the compatibility section.

If iconification cannot be performed, ViNCEd will make the window as small as possible and will move it back behind all other windows.

Left to the iconify gadget are the buttons , at least some can be put there if you don't see any. What these gadgets do is that they invoke a macro as soon as they get pressed, i.e. they are convenient keyboard shortcuts. You may, for example, configure the buttons to "type" the "list" command for you, to see the directory by a press of a button.

To the right and at the bottom of the window are two "scrollers", including a pair of "arrow gadgets". Guess what, they scroll the contents of the window to the left and right, as well as upwards and downwards. As all other gadgets, they can be turned on individually, either with the window path or by providing a default by the SetVNC program, in the third window page .

In case you don't like the shape of the ViNCEd custom gadgets: There is a well-defined interface to replace these gadgets with custom images, which is for example used by the "VisualPrefs" program.

1.20 The Window Buffers

A window buffer is simply the part of ViNCEd that holds the text. Text you've typed and text you see on the screen. Unlike what you might think, there are actually four buffers worth mentioning, not only one.

The first two are easiest to understand, the "upper" and the "lower" display buffer. Both keep the text printed and typed to the window. Whereas the lower buffer is responsible for the text starting at the top edge of the window, the upper buffer keeps all lines "scrolled off" the window - this kind of buffer is also known as "review" buffer, because it allows you to "review" what happened some time ago and was simply scrolled out of the window. Both buffers have a limited size and can only hold a limited amount of lines. As soon as they overflow, the topmost or the bottommost line of the upper resp. the lower buffer are removed to make room for more lines.

This is, if the upper or review buffer is full and the window is scrolled upwards, the topmost line of the lower display buffer is moved to the review buffer, and the topmost = oldest line of the review buffer gets lost. Quite the same happens if the window is scrolled downwards and the lower display buffer cannot hold more lines.

The size of both buffers can be configured either by the settings menu , or with the SetVNC program , the first window page to be precise.

Both buffers hold a bit more than just the plain text. Together with the text, certain "attributes" are kept. They include not only the color and style of the text, i.e. whether the characters are black or blue, italic or bold, but also an attribute that tells ViNCEd whether you typed the text yourself or the text was printed by program. The first kind of text can be re-entered by simply placing the cursor at the line containing the desired input and by pressing the return key. The second kind is "usually" ignored by the return key.

Why this? Simply because you don't want to enter the shell "prompt" as a command, won't you?

The display buffer can also be loaded from or saved to the file, that is done either by the project menu or by the SetVNC program using the GET and PUT arguments.

Text within the display buffers can be marked by "dragging" with the mouse, i.e. place the mouse pointer at the beginning of the area to be marked, then press the left button, then open a text region by moving the mouse while holding the button. The text within the region will appear in reverse color. It can be copied to or cut out to the clipboard for later use. Details are found in the Edit Menu section and in the block section .

The third buffer is called the history . It keeps your inputs, too, but in a different way. The text in the display buffers is mixed input and output, with all the formatting and style information needed, whereas the history keeps simply your inputs, no style data, no formatting information. It cannot be "scrolled" into the window or made visible like the two display buffers before, but is nevertheless useful. Special keyboard functions are available that reprint a line from the history and let you edit it again, or that search for a specific line in the history. It's discussed in detail in the history section.

As for the display buffers, it has a limited size which can be setup with the settings menu or with the first window page of the SetVNC program. It can also saved to or loaded from a file, again with the project menu or with SetVNC , using the GET or PUT shell arguments . There is also a script available that will dump the history to the screen.

The fourth buffer is the "directory cache" of ViNCEd. It keeps the contents of directories searched with the Tab Expansion . It has also a limited size, which is counted in directories to be cached, but it can be setup only with the second window page of the SetVNC preference editor.

1.21 The ViNCEd Menu

Unless you provided certain options in the window path of the ViNCEd window, or another program installed a custom menu, ViNCEd windows will come with a default menu attached to them:

The Project menu

The Edit menu

The Macros menu

The Settings menu

1.22 The Project Menu

The project menu is used to open a new ViNCEd window, to load and save the various buffer of ViNCEd, to run the settings editor, the online-help and other various arrangements of the window itself.

New Window

Invokes the "New Window" macro to open a new ViNCEd window. This macro can be setup on the third system page of the SetVNC preferences editor .

Open...

Loads the display buffer from a file.

Open History...

Loads the command history from a file.

Save As...

Saves the display buffer to a file. The file will be a plain text file with certain enclosed ANSI CSI sequences that encode the required formatting information.

Save History...

Saves the command history to a plain ASCII text file.

Expand Window

Will resize and reposition the window such that it appears as the topmost window with the maximal possible size, possibly leaving out the screen drag bar.

Shrink Window

Minimizes the window to the least possible size and places it behind all other windows.

Next Screen to Front

Identical to the standard LAmiga m keyboard short cut, this menu item will bring the next available screen to front.

Jump to Next Screen

Re-opens the window on the next available public screen.

Help...

Invokes the "Get Help" macro to present the online manual, i.e. this guide. The specific macro can be setup on the third system page of the SetVNC program.

Settings...

Invokes the "Edit Settings" macro to run the SetVNC program. The macro can be setup on the forth system page .

About ViNCEd

Presents the version information and the author of the localization.

Close Window

Invokes the "Quit Shell" or "Quit Program" macro , whichever applies, or sends an "End of File" if the requested macro is empty. Both macros can be setup on the third system page . This menu item is identical to the Generate EOF keyboard function .

1.23 The Edit Menu

The Edit menu provides functions to cut, copy and paste blocks of text to and from the clipboard, functions to search in the command history and functions to clear or reset the window.

Cut

Removes a marked region from the window and places it into the clipboard.

Copy

Copies the marked region into the clipboard and un-mark, i.e. "hide", the region.

ViNCEd can be told to run this operation automatically - without hiding the block - as soon as you release the mouse button when opening a region. This is a bit special for the Amiga, but rather common on Unix (X11) machines.

The "Implicit copy after text marking" flag, how it's called, can be found on the second edit page of the SetVNC program.

There is a second flag that changes the operation of "Copy" and "Cut" operations, namely which part of the text should go into the clipboard: Only your input, or all text, even output. (For details about the difference check the buffer section.)

This flag is called "Don't write printed text into clipboard" and can be found on the same page .

Hide

Remove the block mark , i.e. un-highlite the text.

Paste

Insert the contents of the clipboard as if it was typed in by you. That might be a bit dangerous because anything what's left in the clipboard is interpreted as commands by the shell or any other program receiving input from the ViNCEd window. A "delete #? force all" in the clipboard could be quite desasterous!

A similar operation can be performed without the menu; If you own a three-button mouse, you may insert the clipboard contest with the middle mouse button. It will first place the cursor at the position pointed to by the mouse, and then perform the same operation as "Paste" above.

Using the middle mouse button as "Paste" replacement is, too, not very Amiga like; the feature has been adapted from the Unix world as well.

If you do not want to leave the middle mouse button for this "Paste" operation, you may disable it on the second edit page of the preferences editor .

If you do not own a three button mouse, there is a replacement sequence: Hold the Ctrl key and press the left mouse button. This replacement works even if the use of the middle mouse button has been disabled.

Clear Window

Erases the lower display buffer and requests a new shell prompt at the first line of the window. This is identical to the Clear Screen keyboard function .

Erase to End of Window

Removes all text behind and below the cursor position. Quite the same operation is performed by the Delete End of Display keyboard function .

Reset Terminal

Resets the terminals. The lower display buffer will be cleared, the window style flags and colors will be requested from the globally active settings and the text styles will be reset.

If you press the Shift key while selecting this item, the reset will be a full reset. Additional to the actions taken above above, the command history and the upper display buffer will be erased as well.

The unshifted variant is identically to the keyboard function Reset , the shifted full version is the function Full Reset .

Search Forwards

Search the history in "upwards" direction. This is identically to the Search History Upwards keyboard function .

Search Backwards

Search the history in the "downwards" direction. Identical to the Search History Downwards keyboard function .

1.24 The Macros Menu

The macros menu provides items to invoke macros , as well as to define them and to place them into this menu. A macro, for short, is a shortcut for a longer keyboard sequence you could have typed in as well. For example, a macro could type in the letters l i s t Return to show the contents of the current directory with the press of a button.

More details about macros can be found in the macro section .

The macros menu consists first of the macros itself, with the keyboard equivalences RAmiga 0 to RAmiga 9. Selecting any of these items will just run the macro that is shown in the menu. In the case the macro is too long to fit in, only the first few characters will be shown there, followed by an ellipsis "...".

The first item, however, allows you to define macros: First, type the keyboard macro anywhere on the screen, wherever you like. For the example above, you had to type "list".

Then use the mouse pointer and mark the macro body in a block . If you want to include a final Return character as well, mark the line end together with the macro text, i.e. drag the mouse over the end of the line; ViNCEd will now show a dotted half block at the end of the line which indicated that the line end is part of the block.

As soon as you've captured the macro in a block , select the macro you want to replace FROM THE FIRST "Cut Macro" sub-menu. ViNCEd will now remove the macro text from the screen and insert it into the menu at the desired position. By the way: This sub menu will not be available unless you really marked a block, of course...

The marked line end will appear there as "\r", which is the ViNCEd way of saying that the Return key should be pressed. More about these "macro control sequences" can be found in the macro section .

1.25 The Settings Menu

As you might have guessed, this menu covers some frequently used settings of ViNCEd. There are far too many possible settings to fit in this menu, all of them can be selected with the SetVNC program , so I had to make an decision and made only the most frequently used available in this menu.

The first three items in this menu are used to set the size of the various buffers ViNCEd provides, namely the size of the command history in lines, the size of the lower display buffer in lines and the size of the upper review buffer , also given in lines. To set any of these sizes, type in the desired buffer size, wherever you like, capture it in a block and select the appropriate item.

The following items toggle-select various flags ViNCEd offers, please follow the links:

Dos Cursor Mode

Overwrite Mode

Wrap Around Buffer

Smart Close

Safer Close

Cut Inputs Only

Rebuild Delay

Auto Copy

NumPad for Cursor

1.26 Overwrite Mode

If this menu item is checked, the overwrite mode is active. This means that text you type overwrites the text that is already in the window, it is not inserted as usual. The TAB Forwards keyboard function behaves then a bit different, too. Instead of inserting as many spaces as required to place the cursor at the next tabulator stop, the cursor gets just moved. Quite the same happens to TAB Backwards .

There is also a keyboard function that toggles this flag; it is called Toggle Overwrite and is bound in the default setting to Alt Num0.

1.27 NumPad for Cursor

If the "NumPad for Cursor" menu item is checked, ViNCEd will set the "NumLock" qualifier. This qualifier changes the behavour of certain keys in the keyboard, quite similar to the Caps Lock key with the exception that there is unfortunately no light to indicate its state.

In the default keyboard setting, this flag will turn the numbers on the numeric keypad into cursor key equivalents, i.e. they can be used to move the cursor around.

This menu item is functional identically to the Toggle NumLock keyboard function ; it is bound to the keyboard combination Alt Num[in the default keyboard configuration.

1.28 Block Operations

A "block" is, in ViNCEd terms, a "marked" or "hilited" area of text. With the help of the edit menu , certain operations can be performed with the text in the block. It can be copied to the clipboard and by this made available to other programs, it can be cut out, and the clipboard contents can be re-inserted back into the window.

The text in the "block" is printed with reverse colors, but the block may contain something more "not so obvious", the "marked line end". It will be shown as a checkered box at the end of a line, of about half the size of the cursor. The marked line end is a special character you are not able to see except for its function and when it's marked. It keeps lines "apart" and marks the end of a line on the display. This is somehow the printed equivalent of the Return key on the keyboard - not a letter, but still something important. If the text block is later on re-inserted from the clipboard, this marked line end is substitutes precisely by a press to the Return key.

To mark a block, first move the mouse pointer to the beginning of the desired block, probably use the scrollers to make the desired part of the display buffer visible, then hold the left mouse button. While holding the mouse button, move the mouse to the end of the desired block, then release it. This is referred to as "dragging". If the block is too large to fit on one window, move the mouse pointer to the edges of the window to make ViNCEd scrolling the contents.

If you're marking more than one line, the "ends" of the lines in between will become marked automatically. However, if the line end of a single line should be marked, the mouse pointer must be moved behind the line explicitly until the checkered box appears.

More about the clipboard functions is in the edit menu sections .

Just as a side remark: ViNCEd DOES NOT require the ConClip program. If you're working exclusively with ViNCEd, you may remove this program from your startup-sequence.

1.29 Macros and Buttons

Macros and buttons are used for quite the same: They replace lengthy keyboard sequences. The only way in which they differ is how they are defined and how they are "invoked". "Invoking" means that the "contents" of the macro and the button - which is plain ASCII text - is put into the keyboard input buffer, as if you had typed this text yourself. Because not all keys are represented by a printable ASCII character, certain "control sequences" are available that expand into these keys - more on that below.

Macros are defined by either the macros menu by cutting them directly from the text, or by the preferences editor, SetVNC , on the Macros pages. Macros are made available thru the macros menu and the keyboard shortcuts RAmiga 0 to RAmiga 9.

Unlike macros, which consist only of their "macro body", the buttons contain an additional string, the "button title". This button title will appear in the title of the ViNCEd window and should be as short as possible, to leave room for many buttons. Buttons are invoked by their gadgets in the window title, and can be defined exclusively thru the SetVNC program, at the third macro page . Like macros, buttons have keyboard shortcuts - the function keys on the keyboard. The F1 key invokes the rightmost button, F2 the next button to the left and so on.

Now for the special characters in the macro body. This string is "almost" a C string with the standard C syntax for special control characters.

A plain ASCII character is hereby represented by itself, special characters are "escaped" with the backslash "\" and an ordinary character - this sequence expands then into a control character. The backslash must be, therefore, escaped with another backslash. Here's the list:

\t: The TAB character, ASCII 09 \b: The Backspace character, ASCII 08 \r: The CR character, ASCII 0D. This is the code send by the Return key. This sequence MUST be used to include a press to Return in the macro body. \n: The LF character, ASCII 0A. Do not mix this with \r. \a: The BEL character, ASCII 07 \f: The FF character, ASCII 0C \\: The backslash itself. \" The double quote \' The single quote \[The opening square bracket \] The closing square bracket.

Additionally, there is a way to insert a character by its ASCII value: Type the backslash \, followed by the ASCII value, followed by a dot (.). The ASCII value can be either given as a decimal number, or, if preceded by $ or 0x, as a hexadecimal value, with § as octal and with % as binary number. The # marks the number as decimal explicitly. Several examples follow:

\155. The CSI character 9B in decimal \0x9b. same in hex, \$9b. again in hex, but slightly different notation \§233. and in octal

Please note the "." at the end of the number. This makes some difference, as for example

\$9B. the CSI character, and \$9.B the ASCII TAB, followed by the character B

is quite different. By using the CSI, all CSI keyboard sequences ViNCEd understands can be inserted into the macro, for example:

\$9B.A move the cursor upwards one line

The complete list of CSI keyboard sequences is simply too long to presented here, it's described in detail elsewhere .

The single or double quotes must be used to enclose leading or trailing spaces that would be usually ignored:

" " the plane space itself.

The square brackets have a special meaning. Whenever they appear, the string between "[" and "]" is considered an Amiga DOS file name pattern and a file requester is opened. You're then allowed to pick a filename the square brackets will expand into. For example, consider the following macro:

MultiView [#?.jpg]\r Show my favourite pictures!

Whenever this macro is invoked, a file requester will pop open, allowing you to select a picture - or to be precise, a file whose file name ends with ".jpg". This picture is then shown by "MultiView". Ain't that great?

1.30 The Window Path

The "window path" is for ViNCEd what's the file name for an ordinary disk. This path is, for example, used in the "WINDOW" tooltype of the Shell icon on your workbench and specifies not only that the output of the Shell should go to a console window, but also a lot of parameters for the window.

ViNCEd knows quite a lot of parameters here, including the standard CON: arguments and all parameters of ConMan. Here's the the list:

VNC: [W userwindow/]

[S userscreen/] [B superbitmap/]

leftedge/ topedge/

width/ height/

title [/COLS cols]

[/ROWS rows] [/WAIT]

[/AUTO] [/CLOSE]

[/NOCLOSE] [/SMART]

[/SIMPLE] [/INACTIVE]

[/BACKDROP] [/BACK]

[/NOBORDER] [/SIZE]

[/NOSIZE] [/DRAG]

[/NODRAG] [/DEPTH]

[/NODEPTH] [/NOMENU]

[/MENU] [/NOPROPX]

[/NOPROPY] [/PROPX]

[/PROPY] [/FALLBACK]

[/NOFALLBACK] [/OLDLOOK]

[/CHUNKY] [/PLANAR]

[/SHELL] [/NOSHELL]

[/BUTTONS] [/NOBUTTONS]

[/ICONIFY] [/NOICONIFY]

[/ICONIFIED] [/ANSI]

[/NOANSI] [/WINDOW window]

[/FONT name.size] [/KEEP]

[/SCREEN pubname] [/ALT leftedge/topedge/width/height]

[/STITLE title] [/SDEPTH depth]

[/SFONT name.size] [/TITLEBAR]

[/NOTITLEBAR] [/MONITORID id]

[/MONITOR name] [/PLAIN]

[/PREFS pathname] [/conmanoptions]

All options in square brackets are, uhm, optional. Quite a lot of choice, don't you think so? Even some of the non-optional arguments, like the position arguments "leftedge", "topedge", "width" and "height" can be left blank. ViNCEd picks reasonable defaults in this case. The only thing that MUST NOT be dropped is the name of the handler , VNC: in the line above. If this is the only argument given, and only in this case, ViNCEd will get its path from a different location, namely from the "default path" which can be setup with the SetVNC program on the forth system page . It uses the same syntax except that the leading "VNC:" - the name of the handler - MUST NOT be given in the preferences.

And finally, another note here about proper "escaping":

If you need to put anywhere in this string a literal forwards slash that should be regarded as character, not as argument separator, either escape it with a backslash, i.e. write \/, or enclose the complete parameter of the argument in double quotes, as for example

..../SCREEN"My/Screen"... identical to .../SCREENMy\/Screen...

to put the window on a public screen of the (undoubtfully silly) name My/Screen. To insert a literal double quote or a literal backslash, just escape with a backslash. That is

\\ is the backslash, literally.

\/ is the forwards slash, but only outside of double quotes

\" is the double quote

Finally, if you want to put all this into a shell command line, all double quotes in a double-quoted string must be escaped with an asterisk again, because the shell parses this string, too. That is, you need to write *" to get a single double quote within a quoted string in a shell argument. Silly? Guess so... An even sillier (and admittedly ill) example is discussed in the window title section.

1.31 The name of the ViNCEd handler

The name of the ViNCEd window handler chosen by the installation script is "VNC:", or "CON:" if you choose to replace the default console handler by ViNCEd; this replacement is done by the SetVNC program which is called in the User-Startup on startup.

Another possible name is, for backwards compatibility, "NEWCON:".

Please DO NOT mount ViNCEd under different names, the fixed names are required for the tricky internal mounting algorithm; the problem is here that ViNCEd is both, a library and a handler. Unlike all other handlers, it can be flushed from memory if it is no longer required.

A third available name is "VNR:". This is reserved by the installer script for the RAW version of ViNCEd.

1.32 ConMan W Argument

This ConMan compatible argument is used to install a ViNCEd handler into an already existing intuition window. The address of the intuition window structure must be given as a hex number following the ’W’, optionally with a leading ’$’ or ’0x’. The intuition window will be closed when ViNCEd is done unless the KEEP parameter is given.

This parameter is exclusively reserved for the experts, don’t play with it.

1.33 ConMan S Argument

This ConMan compatible argument selects an already existing intuition screen ViNCEd will open its window on. The screen is supposed to be a private custom screen and must stay open until ViNCEd closes its window. The address of the custom screen must be given as a hex number following the ’S’, optionally with a leading ’$’ or ’0x’.

This parameter is exclusively the experts. If you want to open ViNCEd on its own screen, use one of the parameters SDEPTH , SFONT , MONITORID , MONITOR , TITLEBAR or NOTITLEBAR .

1.34 ConMan B Argument

This ConMan compatible argument will turn the ViNCEd window into a SuperBitmap window, with a bitmap allocated by the invoking program. The address of the bitmap must be given as a hex number following the ’B’, optionally with a leading ’$’ or ’0x’. The bitmap structure WILL NOT be released when the window is closed.

This parameter is here just for backwards compatibility, its use is not recommended. Superbitmap windows are usually much slower than the standard "simple" or "smart" refresh windows ViNCEd uses. Furthermore, leave it to the professionals to play with this argument.

1.35 LeftEdge Argument

The left edge given in pixels from the left edge of the screen the window will appear on. ViNCEd may try to move or scale the window to make it fit on the screen, the window may appear for this reason at a slightly different position.

You may drop this argument, ViNCEd will use some reasonable default.

1.36 TopEdge Argument

The top edge of the ViNCEd window to open, counted in pixels from the left edge of the screen. ViNCEd may adjust this value to make the window fit on the screen.

If you select a top edge of "-1", ViNCEd will adjust the top edge of the window to the bottom edge of the screen drag bar.

If you drop this argument, ViNCEd will choose a reasonable default.

1.37 Width Argument

The width of the ViNCEd window to open, in pixels. ViNCEd may adjust this value to make the window fit on the screen.

If the width argument is "-1", the window will be made as wide as possible.

If you drop this argument, ViNCEd will select a reasonable default for you.

Please note that there is also the COLS argument which selects the size of the window in characters of the selected window font instead using the size in pixels.

1.38 Height Argument

The height of the ViNCEd window to open, in pixels. ViNCEd may adjust this value to ake the window fit on the screen.

If the height argument is "-1", the window will be made as wide as possible.

If the argument is dropped, ViNCEd will select a reasonable default for the width.

If you want to specify the height in text characters of the selected font instead in pixels, use the ROWS argument.

1.39 COLS Argument

Specifies the width of a ViNCEd window in columns of text of the selected font that should fit into the window. The number must follow the COLS argument and can be given in decimal, or hexadecimal with a leading '$' or '0x'.

ViNCEd may adjust this value to make the window fit on the screen.

1.40 ROWS Argument

Specifies the height of the window in rows of text of the selected font that should fit into the window. The number of rows must follow ROWS, and can be given in decimal or hex, with a leading '$' or '0x'.

ViNCEd may adjust this value to make the window fit on the screen.

1.41 WAIT Argument

If given, ViNCEd will keep its window open even if the last program that used this window closed its stream to ViNCEd. The window must then be closed by the user explicitly, either by pressing the close gadget or by using the keyboard EOF function , which is usually bound to Ctrl \.

1.42 AUTO Argument

If given, ViNCEd will close the window temporary if the user pressed the close gadget and no program is waiting for input. The window will pop-open again as soon as some program running in this window requests input or prints into the window.

The behavour of ViNCEd if more than one program uses this window can be setup with the Don't send EOF until everybody waits flag.

1.43 CLOSE Argument

If given, ViNCEd will add a close gadget to the window. This is, however, the ViNCEd default since the very first release; there was always a close gadget...

Anyways, if you don't like this default, you may turn it off on the third window page of SetVNC .

1.44 NOCLOSE Argument

If given, ViNCEd will not add a close gadget to the window and hence possibly override its default to add one.

The default can be setup on the third window page of SetVNC .

1.45 SMART Argument

If given, ViNCEd will use a "smart refresh" window. This means that the window refresh will be done by the operating system, not by ViNCEd itself. The resulting window might refresh a bit faster therefore, but will also use more chip memory for the buffer. The most important advantage of "smart refresh" windows is, however, that the scrolling will be a bit smoother.

The default is SIMPLE .

1.46 SIMPLE Argument

Tells ViNCEd to use a "simple refresh" window. This means that ViNCEd is responsible for reprinting the window contents in case part of the window gets damaged by overlaying windows. Since ViNCEd is rather fast in printing and this option is less memory extensive than SMART , this is the default.

1.47 INACTIVE Argument

If given, ViNCEd will not activate this window automatically on open, i.e. the input focus will not change.

1.48 BACKDROP Argument

This argument will turn the window into a backdrop window which sits always behind all other windows and cannot be brought to front. The main usage of this argument is to make ViNCEd appear on its own screen, without any "visible window" or border.

To make this working, specify the options NOBORDER , NOSIZE , NOPROPX , NOPROPY , NODRAG , NODEPTH and NOBUTTONS as well, select an empty string as title and adjust the position and the size of the window accordingly. These options are easely forgotten and therefore bundled in the "meta"-option BACK . With all that selected, use the SDEPTH option to open a custom screen for ViNCEd.

The result will be a "full screen" ViNCEd which looks like a terminal.

1.49 BACK Argument

This argument will turn the window into a backdrop window which sits always behind all other windows and cannot be brought to front; it will also suppress the window border and all buttons and sliders in it, as there are the close gadget, the drag gadgets, the horizontal and vertical sliders and much more. The main usage of this argument is to make ViNCEd appear on its own screen, without any "visible window" or border.

If you specify the SDEPTH option as well, ViNCEd will open on a custom screen as a "shell screen", almost like a terminal.

1.50 NOBORDER Argument

If specified, ViNCEd will tell the operating system not to draw a frame around the border. The result will be usually ugly looking because all the other system gadgets will still be drawn unless you turn them off explicitly. The main use of this option is to make ViNCEd appear on its own screen without a visible window border. Check the BACKDROP or BACK option for details about this trick.

1.51 SIZE Argument

If present, ViNCEd will add a sizing gadget to the window. Since this is the default, this argument does usually not much and is just provided for symmetry.

1.52 NOSIZE Argument

If present, ViNCEd will not add a sizing gadget to the window.

1.53 DRAG Argument

If present, ViNCEd will provide a drag bar for the window, i.e. the window will be movable. This is the default, hence this option does usually not very much. It is just provided for symmetry.

1.54 NODRAG Argument

If present, ViNCEd will not provide a drag bar for the window, i.e. the window will not be movable. This is seldom useful except for building windows without any actual border at all. For details, check the BACKDROP documentation.

1.55 DEPTH Argument

If given, ViNCEd will install a depth arrangement gadget in its window, i.e. you may bring this window in front of and behind other windows. Since this is the default anyways, this argument does usually nothing useful. It is just provided for symmetry.

1.56 NODEPTH Argument

If given, ViNCEd will not install a depth arrangement gadget in its window. This is seldom useful, possibly except for building a window without any visible frame at all. For details, check the BACKDROP argument.

1.57 NOMENU Argument

If present, ViNCEd will not attach its own menu to the screen. However, most block functions like "Cut", "Copy" and "Paste" will be still available by their shortcuts, provided you bound these shortcuts to the proper keys.

The default is to add a menu for windows running in the shell mode , and not to add a window to all other windows.

1.58 MENU Argument

If present, ViNCEd will attach its menu to the window.

The menu will be there anyways provided the window was opened in shell mode , but must be requested explicitly for all other windows.

1.59 NOPROPX Argument

If this argument is given, ViNCEd will not add the horizontal slider at the bottom of the window.

This slider can be disabled by default as well, check the third window page of the SetVNC program.

1.60 NOPROPY Argument

If this argument is given, ViNCEd will not provide the vertical slider at the right edge of the window.

This slider can be disabled by default as well, check the third window page of the SetVNC program.

1.61 PROPX Argument

If given, ViNCEd will add a horizontal slider to the bottom edge of the window if this slider was disabled on the third window page of the SetVNC program.

1.62 PROPY Argument

If given, ViNCEd will add a vertical slider to the right edge of the window if this slider was disabled on the third window page of the SetVNC program.

1.63 FALLBACK Argument

If this argument is given and ViNCEd is told to open on a public screen which is not available, ViNCEd will fall back to the default public screen.

This is also the default behavour.

1.64 NOFALLBACK Argument

If this argument is present and ViNCEd is told to open on a non-available public screen, it will fail and refuse to open at all.

1.65 OLDLOOK Argument

If given, ViNCEd will color the menus in the old pre-3.0 way, whatever this might be good for.

1.66 CHUNKY Argument

Tell ViNCEd explicitly not to try to optimize scrolling for native amiga displays. This optimization effort will eventually slow down the graphics output considerably on gfx boards.

However, ViNCEd will be usually smart enough to detect if a native or a non-native screen is available, so this argument is not strictly required.

The "chunky" mode can be turned on by default as well, it's on the first system page of SetVNC .

1.67 PLANAR Argument

Tells ViNCEd that it the window will appear on a native amiga screen and it should try to optimize the scrolling speed. However, since the PLANAR option is automatically turned off anyways as soon as a "chunky" non-native screen is detected, this option doesn't do too much in the current implementation.

The "chunky" mode can be turned on by default as well, it's on the first system page of SetVNC .

1.68 SHELL Argument

If this is present, the window is opened in shell mode . This has quite a lot of consequences, as turning on the menu, the TAB expansion and other goodies. More details are found in a separate chapter .

The shell mode can be turned on by default as well, the responsible flag is found on the first shell page .

1.69 NOSHELL Argument

If this is present, the window is not opened in shell mode , but in standard mode. This has quite a lot of consequences, as disabling the menu unless turned on explicitly with MENU , disabling the TAB expansion and other goodies.

The default state of the Shell mode can be controlled by SetVNC as well, it's on the first shell page .

1.70 BUTTONS Argument

If given, ViNCEd will add the buttons to the window title and will hence make them available.

This is also the default for windows operating in Shell mode .

1.71 NOBUTTONS Argument

If given, ViNCEd will suppress the button gadgets in the window title.

This is the default for non-shell windows.

1.72 ICONIFY Argument

If given, ViNCEd will add an iconification gadget to the window. If this gadget is pressed, ViNCEd will turn into an icon on the workbench screen, if possible. Output will resume in this case as usual, except that it won't be printed, but is just kept internally.

However, there's a certain quirk with this gadget: Certain programs make it impossible to iconify the window - the window can't be closed safely as soon as these programs are run; the most (un)famous example is the "More" command. There is not much I can do about it except offering a work-around: The installation procedure should have installed a tiny script in your "S:" drawer which replaces the standard "more" and should be used instead. If possible, setup your path in a way such that "S:More" is used instead of the standard more command. If unavoidable, run the command SetVNC FreePointer in the shell -this will allow iconification in most cases. Details about this problem can be found in the compatibility chapter .

The iconification gadget will be added by default, but if you don't like the iconification gadget, you may turn it off with the third window page of SetVNC .

1.73 NOICONIFY Option

If this flag is present in the window path, ViNCEd will not add an iconification gadget to the window. You may also choose to make this the default, there is a flag on the third window page of the preference editor that will disable the iconification gadget unless you ask for it explicitly.

For more insight about this gadget, read the notes about the ICONIFY gadget and the gadget section .

1.74 ICONIFIED Option

If given, ViNCEd will open the new gadget already in iconified state, i.e. as an icon on the workbench. All printing will go into the internal buffers until the icon is double-clicked and hence opened.

There is a tiny problem with this icon, namely that ViNCEd won't be able to adjust the size of the "virtual window" to the real screen size (simply because there is no real window), it will try to "guess" how big the window should be. A program that asks for the window size - this can be done by sending CSI sequences - will therefore receive the size of the window ViNCEd *thinks* it will get when it is opened. It's usually quite good at guessing, though, and that's not a real problem either.

1.75 ANSI Argument

If this argument is present, ViNCEd will use the standard ANSI colors for the text pens in the window instead of simply mapping the text colors directly to pen numbers. A detailed description of the ANSI colors can be found in a separate section .

You may also choose to use the ANSI coloring by default - check for this the third edit page of SetVNC .

1.76 NOANSI Argument

If NOANSI is present in the window path, ViNCEd will assign the text colors directly to pen numbers, as it always was for the CON: windows. For details about what ANSI colors are, check the ANSI colors section of this guide.

A flag of the SetVNC program on the third window page controls the default of this flag, i.e. whether ANSI or NOANSI is the implicit default.

1.77 WINDOW Argument

This argument is similar to the ConMan W argument and allows to attach a ViNCEd handler to an already existing intuition window. The argument takes an additional parameter, the address of the intuition window structure as a hexadecimal number, optionally with a leading '$' or '0x'. Decimal numbers with a leading '#' to indicate the base are welcome, too. The window will be closed as soon as ViNCEd is closed. If you want the window to stay open, then specify KEEP as well. ViNCEd will be "more friendly" to this guest window in this case.

BE WARNED! This option is definitely for the "professional power user" and should not be played with.

1.78 FONT Argument

This argument takes an additional parameter, namely the name of the font and its size ViNCEd shall use for the text in the window. The argument is the base name of the font, a dot, NO trailing "font", but instead the size of the font to be used as a decimal number.

For example, to use the "topaz.font", size 9, use an argument like this:

FONTtopaz.9

Similarly, FONTcourier.13 will use the "courier.font", size 13.

NOTE: This will ONLY set the font for the text within the window. ViNCEd will use the font of the screen it appears in for its menus and the title bar. If you open ViNCEd on its own screen, use SFONT to set this additional font.

Another NOTE: DO NOT use proportional fonts. They will produce certain graphic artifacts on the window.

A third note: By an unfortune error, the popular XEN font, size 8, is a proportional font. Please use the fixed "XEN" font in this archive instead.

1.79 KEEP Argument

This argument is only relevant if a ViNCEd handler was installed into an already existing window with the WINDOW or the ConMan W argument. Hence, this is somewhat for the professional user.

If KEEP is present, ViNCEd will be somewhat nicer to the user window. It will not try to close the window - the user program has to do that - and will not try to resize it. This is mainly useful if you want to run ViNCEd in a part of your own window your application created.

1.80 SCREEN Argument

This argument tells ViNCEd that it should open its window on a given public screen; the name of the public screen - not the title! - must follow as parameter to the SCREEN argument. For example, to open a window on the public screen named "GOLDED.1", use the argument

SCREENGOLDED.1

If the screen name contains any slashes "/", these must be escaped by a backslash \, or the whole string must be included in double quotes; if white spaces are included in the name of the public screen, it should be quoted with double quotes, too - as in

SCREEN"My Slash/Screen"

Since the standard amiga shell parses this string, too, the double quotes must be escaped as well as part of a shell command. The result may look very ugly, as this example shows:

1.SYS:> newshell WINDOW="VNC:////My window/SCREEN*"My Slash/Screen*""

It uses the "*" as (BCPL) escape character for the first string. Urgh.

Hint: The name of the screen in this example is

My Slash/Screen.

If the screen of the given public name does not exist, or is not public, it depends on a possible NOFALLBACK option what happens. If this option is given, ViNCEd will just fail and refuse to open any window. If the option is NOT given, or a FALLBACK is present, ViNCEd will open the window on the default public screen instead.

The SCREEN argument serves another purpose if other options, as MONITORID or SDEPTH indicate that ViNCEd shall open its window on its own screen. If this is the case, ViNCEd will first try to open its window on a public screen of the given name, and if this fails, will open its own public screen under the same name, with the given options, and open a window on its own screen.

Hence, this will put a ViNCEd custom screen into a ViNCEd public screen if the name does not conflict with another screen.

1.81 ALT Argument

The ALT argument takes four numbers as parameters, separated by slashes. They specify the "alternate" window position which is used if you press the "zoom" gadget.

The arguments are, to be more precise, the "left edge" and "top edge" of the window, given as distances in pixels from the left and top edge of the screen the window appears on, and the "width" and "height" of the window in pixels - in this order.

1.82 STITLE Argument

The STITLE option takes one parameter, a string. This string is printed in the screen title as long as the ViNCEd window is active. Except that, it follows the same syntax rules as the standard window title argument.

1.83 SDEPTH Argument

If this option is present, two things happen:

First, ViNCEd is told to open on its own screen. Second, the depth of the screen will be set to the parameter of this argument. That is,

SDEPTH4

will open ViNCEd on a custom screen which makes 2^4 = 16 colors available.

NOTE: To make this screen a public screen, use the SCREEN argument to provide a name for the public screen.

Other options that make ViNCEd open on its own screen are SFONT , MONITOR , MONITORID , TITLEBAR and NOTITLEBAR .

1.84 SFONT Argument

If this argument is present, two things happen:

First, ViNCEd opens its window on its own custom screen. Second, the font for the menus, the title bar and the screen title will be set to the font given as parameter to this argument.

The font must be given by its base name, a dot, NO "font" extender and the size of the font as decimal number; i.e. an argument like

FONTtopaz.11/SFONTruby.15

will open ViNCEd on its own screen, will use the "topaz.font" size 11 for the text in the window and the "ruby.font", size 15 for the menus and the title bars.

If NO additional FONT argument is present, the SFONT will be used for the window contents as well, but in this case only a fixed width font should be used or ViNCEd will create graphic artifacts. If a FONT is given as well, every font is acceptable as SFONT, even proportional fonts.

If you want this custom screen to become a public screen, provide a public screen name with the SCREEN argument.

Other options that make ViNCEd open on its own screen are SDEPTH , MONITOR , MONITORID , TITLEBAR and NOTITLEBAR .

1.85 NOTITLEBAR Argument

If this argument is present, two things happen:

First, ViNCEd opens its window on its private custom screen, second, the title bar of this screen will be shown in behind any backdrop windows. Since ViNCEd will usually NOT use a backdrop window itself, this won't change too much anyways and makes only sense together with the BACKDROP option.

To make this screen a public screen instead of a custom screen, provide the desired public screen name as parameter to the SCREEN argument.

Other options that make ViNCEd open on its own screen are SFONT , SDEPTH , MONITOR , MONITORID and TITLEBAR .

1.86 TITLEBAR Argument

If this argument is present, two things happen:

First, ViNCEd opens its window on its private custom screen, second, the title bar of this screen will be shown in front of any backdrop windows. Since ViNCEd will usually NOT use a backdrop window itself, and even more, this is the default for screens anyhow, this won't change nothing except putting ViNCEd on its own screen. This argument is simply here for symmetry.

More about what a backdrop window is can be found in the section discussing the BACKDROP option.

To make this screen a public screen instead of a custom screen, provide the desired public screen name as parameter to the SCREEN argument.

Other options that make ViNCEd open on its own screen are SFONT , SDEPTH , MONITOR , MONITORID and NOTITLEBAR .

1.87 MONITORID Argument

If this argument is present, ViNCEd is opened on its own custom screen whose "view mode" = "monitor id" is taken from the hex number following the argument. A leading '$' or '0x' is permitted here to indicate hex notation explicitly.

Hence, the argument

MONITORID0x19000

will use a NTSC Hi-Res screen.

Useful monitor IDs can be found in the system documentation - it depends on your hardware which IDs are available.

The default monitor ID can be setup with the SetVNC preference editor, on the first window page .

To make this screen a public screen instead of a custom screen, provide the desired public screen name as parameter to the SCREEN argument.

Other options that make ViNCEd open on its own screen are SFONT , SDEPTH , MONITOR , TITLEBAR and NOTITLEBAR .

1.88 MONITOR Argument

If this argument is present, ViNCEd is opened on its own custom screen whose "view mode" = "monitor id" is taken from the name of the monitor following the argument.

Hence, the argument

MONITORNTSC:High Res

will use a NTSC Hi-Res screen, at least for the english language as system selected language.

Useful monitor IDs can be found for example by browsing the monitor data base with the system preferences editor "Screen-Mode".

NOTE: Except for private use, I RECOMMEND NOT TO USE this argument. The point is that the monitor names, unlike the monitor IDs, differ from localization to localization. A program written for the english language will, hence, not work for german users and vice versa. Keep care and try the MONITORID option whenever possible.

The default monitor ID can be setup with the SetVNC preference editor, on the first window page .

To make this screen a public screen instead of a custom screen, provide the desired public screen name as parameter to the SCREEN argument.

Other options that make ViNCEd open on its own screen are SFONT , SDEPTH , MONITORID , TITLEBAR and NOTITLEBAR .

1.89 PLAIN Argument

"Degrades" a ViNCEd window back to a console-like window. All ViNCEd specific extensions in the window frame will be disabled, namely the scrollers, the arrows, the buttons and the iconfication gadget. The shell mode will be turned off, too.

It does not, though, disable the CLOSE gadget or its preferences.

1.90 PREFS Argument

This argument provides a different source for the window settings than the system global preferences database. If this argument is used, ViNCEd will read its preferences file - to be written with the SetVNC program and its "SAVE" option - from the file name given as parameter to this option.

Since the path name of this file may contain slashes - which are interpreted as argument separator, these slashes must be escaped by a backslash in front of them. For example, to read the preferences from "SYS:Devpac/ViNCEd.macros", use the following argument:

PREFSSYS:Devpac\/ViNCEd.macros

Things may get even worse if this is used as an argument to a shell command, e.g. for "NewShell". Another level of "escaping" might be necessary, possibly leading to rather absurd constructions. Keep care!

1.91 ConMan style single character options

The following options are single character options provided mainly for backwards compatibility with the "ConMan" console handler. These options should go as the last options on the window path (but may go, unfortunately, anywhere in the path, even though I don't like this...). Interestingly, some official ARexx commands use them. All of these arguments are "toggle switches", which means that giving them twice is as good as giving them not at all. All these characters can be combined into a single string - provided this string does not conflict with any other option as total.

However, I would still not recommend these options. They are mainly provided for compatibility to ARexx and ConMan.

Z Use a "GimmeZeroZero" window. The coordinate origin for drawing in this window is really in the upper left edge of the window contents and not in the upper left edge of the window border. However, ViNCEd doesn't care about these offsets anyways. This makes only sense if you plan to draw in this window. Since this makes re-arrangement of the window slower, this option should be avoided.

B Use a "BACKDROP" window. Check the BACKDROP option for details.

N Use a "BORDERLESS" window. Again, details are in the section discussing the NOBORDER argument.

R Toggle between SIMPLE and SMART windows, the default is SIMPLE. Details are found by following the links.

D Toggles the depth arrangement gadget on or off. Details are again in the section about the NODEPTH option.

M Toggles the window drag bar on or off. Details again under the recommended NODRAG argument.

S Toggles the sizing gadget on or off. Details again found in the NOSIZE section.

C Toggles the close gadget on or off. Again, check the CLOSE and NOCLOSE arguments.

A Toggles whether the window should be activated when it is opened. The default is activation. You should use the newer INACTIVE argument instead.

L A ViNCEd special leftover from releases 1.xx. Toggles the "Luxury switch" on and off. Defaults to on. If off, the window does not get a menu and no custom gadgets, the "Dos Cursor Mode" will be activated. Highly obsolete and likely to be removed.

O Toggles the menu on or off. Default is on for windows using the Shell mode , off otherwise. MENU and NOMENU are recommended replacements.

X Toggles the horizontal scroller on and off. Equivalent to NOPROPX resp. PROPX .

Y Toggles the vertical scroller on and off. Equivalent to NOPROPY resp. PROPY .

G Toggles the buttons in the title bar on and off. Equivalent to BUTTONS resp. NOBUTTONS .

1.92 The Window Title

The window title argument of the ViNCEd window path , and the related STITLE argument are, of course, in first place standard strings. However, this string may contain special control sequences that are expanded as part of the string. You may be able, for example, to display the size of the window, the name of the current directory or the number of the Shell in the window or the screen title. The following list presents all characters that have a special meaning in this string:

" Double quotes should be used to escape leading or trailing blank spaces. While this is not strictly required, it is recommended. It also "brackets" all single forward slashes between the two double quotes, i.e. they are not seen as separators of arguments in the window path . However, read below for special caveats if the title has to be given as an argument to the AmigaShell.

\ The backslash escapes special characters and interprets them literally.

\" The literal double quote, i.e. the double quote as a character, not as a function.

\/ The literal forward slash. If this character should appear in the window title, it MUST be escaped because it's usually read as the argument separator in the window path .

% The command sequence introducer. This character takes a single character as argument. If this sequence is detected, the percent sign and the character is replaced by something else, see below for the list.

%% The percent sign itself.

And now for the list of the available string substitutions with the percent sign. Two types of replacement strings are available. The first type works always, for all ViNCEd windows. The second type requires a shell running in this window and will, therefore require the Shell mode . Further more, these strings are only expanded when the shell is presenting its prompt and is waiting for your input. It would be simply too dangerous to read the shell internal strings at a different time, they may get changed just in the moment when ViNCEd is trying to read them and this won't do good.

Here's now the first list of "harmless" sequences:

%X or %x The width of the window, in characters.

%Y or %y The height of the window, in characters.

%P The name of the public screen this window was opened on, or the title of the screen if the screen is non-public.

%p The name of the public screen if the screen is public, or the string "Default PubScreen" for the default public screen or "Private Screen" for a custom, private screen.

%T The default title of the screen.

%t The screen title of the screen.

%L or %l The current state of the "NumLock" qualifier. If set, "NumL" gets printed, four spaces if reset.

%O or %o The state of the "Overwrite" qualifier. If overwriting is enabled, "Ovwr" is inserted.

%V or %v The current ViNCEd version identifier.

In almost all cases, %T is identical to %t. At least, I haven't seen a counter example yet.

The next strings expand only in the Shell mode , and only if the shell is printing its prompt on the command line and waiting for inputs.

%N or %n The CLI number of the shell running in the window.

%S or %s The current directory of the active shell.

%R or %r The result code of the last command.

%E or %e The secondary result code (the error code) of the last command used.

%F A text version of the secondary result code of the last command. If the last command returned successful and the error code is zero, %F will expand to "no error".

%f Similar to %F except that it expands into the empty string in case the last command did not return an error.

Finally, a remark about using these sequences from the shell. The AmigaShell parses all strings again, in a first step. The parsed string is then sent to ViNCEd, which parses it a second time. That means for you that you have to escape certain control sequences twice, once for the shell and at a second level for ViNCEd. The result might look rather complicated and more like a random line noise than anything useful. For example, the following command

1.SYS:> NEWSHELL "VNC:////*"This is / a *"test*" \\.*""

results in a window with the title

This is / a "test" \

This is because you've to escape all double quotes with the BCPL escape character, the "*", again for the shell. All single forward quotes in the title are enclosed in a pair of double quotes, which are again escaped for the shell with an asterisk. That is, the forward quote in the string stands for itself. The double quotes around the word "test" must be escaped twice because, for first, the shell should regard them as literal characters - that is again what the star is used for - and ViNCEd should read them as literal double quotes, too. That is what the backslash is used for.

Confused? I bet you are...

And finally, I had to type this in an AmigaGuide document which means that the string gets actually parsed THREE TIMES, so I had to escape all backslashes with another backslash in this manual.... that is, what's actually printed there in this manual is

1.SYS:> NEWSHELL "VNC:////*"This is / a \\*"test\\*" \\\\.*""

where I had to put extra backspaces to make it visible for you right now and.....

***Break

1.Python, Monty:> STOP IT! THAT'S GETTING SILLY.

And, ooops, if you see too many backslashes here you're either using an old version of the AmigaGuide instead of MultiView, or it's time to visit your eye doctor... (-;

***Break

1.Python, Monty:> OUT! ALL OF YOU! OUT! NO MORE BACKSLASHES! OUT!

1.93 The Shell Mode

The power of ViNCEd is the "Shell Mode"; without it, ViNCEd will behave "just as a console handler with a full screen buffer". If it is enabled, however, ViNCEd will provide additional functions which are especially useful for a shell running in a ViNCEd window, as there are the TAB Expansion , the icon drop function, the Ctrl-Z function and others. All of them are useful extensions the ordinary "Amiga Shell" does, unfortunately, not provide and hence have been implemented in the console handler instead. Weird....

To enable these features, you've to tell ViNCEd explicitly that a shell is running in its window, so to allow it to extract the information it needs for these functions. To do so, you must provide the SHELL argument in the window path .

If you're using ViNCEd explicitly as console handler for the shell, you might enable this flag by default, hence, make this argument superfluous. This is done by the preferences editor , on the first shell page .

However, please note that there is a special danger to use the shell mode in a window no shell is running in. Nothing bad *should happen* as I implemented certain security precautions, but who knows? This flag should be turned on only and only if you can make sure that ViNCEd isn't used for anything but the shell.

More about the shell mode is in these sections:

Icon drop in the Shell mode

TAB expansion in the Shell mode

All TAB expansion settings at once

The magic Ctrl Z key in the Shell mode

1.94 Icon drop in the Shell mode

If the Shell mode is enabled, you may drag icons from the Workbench screen into the ViNCEd window. ViNCEd will, in this case, insert the file name, the complete path or the directory name of the icon dropped. This is mainly useful if you need to locate a file deep down in the directory tree, which is already available on the workbench. ViNCEd will type that file name for you, even including double quotes if the file name contains blank spaces.

If you want ViNCEd to insert something different than the file name, hold special keys from the keyboard while dragging the icon; the default "qualifiers" selected for you are as follows:

no qualifier : insert the complete path of the icon

Either Alt key : insert only the file name of the icon

Either Shift key : insert only the directory of the icon dropped

However, you may select these "qualifiers" yourself as well if the defaults don't please you. This works, once again, with the SetVNC preferences editor, on the fifth Shell page .

1.95 TAB expansion in the Shell mode

TAB expansion is a feature offered by most Unix shells. It is a very convenient feature that avoids typing long and complicated file names; instead, you just provide a template of the file name and ask ViNCEd for possible "candidates", or "expansions" how they will be called in what follows. Because this feature is particularly useful for the shell, TAB expansion is only available in the Shell mode .

The next lines present a tutorial you should work thru if you're not familiar with ViNCEd's capabilities. (Really, I mean it!)

In case you're just looking for the complete set of configurable settings, they are elsewhere .

ViNCEd offers six configurable TAB expansion keyboard functions , each of them as "forwards" and "backwards" moving function - it will become clear in a minute what this means. Each of these functions can be bound to a keyboard combination of your choice, and can be configured individually as well.

The default configuration - and that is what is going to be discussed here as an example - uses the Ctrl TAB key for "forwards" and the Shift Ctrl TAB key for "backwards" expansion. Additional keyboard bindings exist to the other functions, but they work in a similar manner. Use the SetVNC keyboard pages adjust the key bindings if these defaults do not please you.

What happens now if you press Ctrl TAB?

In short words, ViNCEd reads the shell argument under the cursor, treads it as an incomplete file name and searches for possible expansions, in locations setup by your configuration explicitly; these include the current directory, the directories in the shell "command path", the "C:" assignment, the list of devices and the list of resident commands.

Which of these sources is searched depends on a set of priorities you assigned to them, and on explicit "hints" ViNCEd takes from the argument to be expanded. As for example, if the argument ends with a colon ":", only the device list will be searched, regardless of the priorities.

The found matches are sorted by the priority you assigned to the different file types. Found expansions are then inserted into the argument again, possibly showing a file requester. Once an expansion is complete, you may use the TAB expansion key again to traverse the list of found matches. The "forwards" function moves in this list in the one, the corresponding "backwards" function in the opposite direction, showing the completions one after another.

Since quite a lot of settings involve the procedure of the TAB expansion and giving the complete list of settings involved might be more confusing than helping, I'd like to present this in the form of a tutorial. As soon as more details are required, follow the links. This tutorial assumes that you're working with the default settings, in a ViNCEd window with the Shell mode enabled.

Go to the root level of your hard disk and enter the following line in the shell, (without the prompt, obviously...)

1.SYS:> rex

The colored box in this line represents the cursor. Press now the TAB expansion key Ctrl TAB.

The line above tells ViNCEd to search for commands that start with rex . ViNCEd will search therefore the current directory, the C: assignment and the complete shell path, i.e. the directories which are printed by the "Path" command. This can be changed , of course.

On my system, this template will match the "Rexx" directory on the root level of my hard disk, the "RexxMast" program in the "Systems" folder which is in the command search path , the "REXX:" assignment , and the icon file "rexx.info".

More details on how to adjust what should match and what not can be found by following the links above, they will explain how to exclude certain matches.

To continue with the example, ViNCEd will answer with the following line:

1.SYS:> Rexx

The template has been refined to "Rexx", but the cursor is placed directly behind the "x", without an additional blank space, allowing you to continue typing. This means the following:

- ViNCEd found some matches. It would have flashed the screen if it would not.

- All the matches start with the letters "Rexx". That is why ViNCEd was able to present a refinement . If you don't want a refinement in that case, you may turn that feature off and ask ViNCEd to insert the first possible match instead.

- However, ViNCEd was not able to find a unique extension, i.e. there is more than one file that matches the template . This is indicated by placing the cursor directly behind the "x". If a literal "Rexx" would have been the only possible match, a blank space would have been inserted. * Additionally, you may ask ViNCEd to show a requester in the case more than one match, i.e. no unique match was found.

Now wait for a second or two, and press Ctrl TAB again. ViNCEd will present now the first match in its full form, the Rexx directory:

1.SYS:> Rexx/

Since this is a directory, a forwards slash "/" instead of a blank space is inserted behind the name.

You've just seen another purpose of the TAB expansion functions: As soon as a list of all matches has been constructed, these functions are used to view this list "item by item". As for the command history , you may move in this list in two directions - and that is why all TAB expansion functions come in two kinds: Both kinds use the same settings, but as soon as the list of matches is constructed, one function moves in "forwards" direction whereas the other moves in backwards direction. For the default settings, the forwards-moving function is bound to the Ctrl TAB key you've just used, its backwards moving counterpart is Ctrl Shift TAB.

The items on this list are sorted by a "priority" you assign to possible candidates. It's the purpose of the SetVNC program to setup these priorities - they can be found on the fourth shell page , one set of priorities for each pair of the six assignable TAB expansion functions.

Pressing now Ctrl TAB again shows the next match:

1.SYS:> Rexx.info

It is this time the icon file "Rexx.info". Since this is a file anyhow, a space is behind it, as show above.

If you don't want to see these files, you may exclude them explicitly , or may at least lower their priority - they will appear then at a later time behind more important files.

The next match - press Ctrl TAB again - is:

1.SYS:> RexxMast

...the RexxMast program. This program is usually not located at the root level of the system disk, but in the "System" directory. The reason why it was found anyhow is that you expanded the first argument on the shell line. ViNCEd noticed that and considered that you may look for a command. Since the Shell looks for commands in the current directory, the C: assign and the "command path", so does ViNCEd. The "System" directory is in most installations part of the "path".

If you don't want ViNCEd to search the complete path, you may simply adjust some priorities of the configuration .

Let's check the next match - press Ctrl TAB again:

1.SYS:> REXX:

It's this time the REXX: assign. If you don't want to see assignments: Nothing as simple as this, the answer is again: Adjust the priorities.

By the way: Didn't you wonder why "RexxMast.info" wasn't found? ViNCEd notices that this file CAN'T be a command, and hence you're surely not looking for non-commands in the command search path. The reason why "Rexx.info" WAS found, but "RexxMast.info" not is, that the first file is in the current directory - might be of some interest - whereas the second is not. In the same spirit as icon files never match outside of the current directory, neither do directories.

Now let's continue - press again Ctrl TAB.

1.SYS:> REXX:

...and just a flashing display. That's the end of the list, no more matches have been found. Things behave a little different if the Wrap Around Buffer in the settings menu was turned on. In this case, ViNCEd will insert a blank line - to indicate that the end of the buffer was reached. The next Ctrl TAB will re-insert the first match of the list.

We can now go backwards from the end of the list to the beginning with the twin "backwards" function - which is in the default configuration the Ctrl Shift TAB key. You'll see

1.SYS:> RexxMast

the "RexxMast" program again.

Use now the backspace key to erase the expansion up to the "Rex" we started with. This will also abort the expansion, as well as any other key except the TAB expansion functions . Press now Ctrl TAB again to re-run the expansion. As soon as the refined template shows up, press Ctrl TAB again. This will now show a requester with all the matches found, even including the assigns and the commands not in the current directory. This is due to some "ViNCEd magic" you may explicitly turn off . However, dependent on the requester package you're using - asl or reqtools - you may or may not see any icons, i.e. ".info" files because they are possibly filtered out .

If you now pick an entry from this requester and click "O.K." in the requester, this entry will be inserted in the ViNCEd window for you. Using this requester, you may even enter the found directories or assigns and select a file from there.

However, if you don't like this "double TAB requester" or it's "in the way", you're of course free to turn it off .

By the way - the time delay ViNCEd allows to occur the two TABs to happen within is just the "mouse double click delay" you may adjust with the "Input" system preferences editor.

Just a last note: Since there are actually SIX tab expansion functions, each of them in two kinds - forwards and a backwards function - and each of these functions can be bound to a keyboard key of your choice and configured individually, there's I hope a configuration that everyone pleases and a TAB expansion for every situation...

A complete overview of all the TAB expansion settings is available too.

1.96 All TAB expansion settings at once

Most of the TAB expansion settings are under control of the third and fourth shell page of the SetVNC program . However, which key is bound to which of the six keyboard functions that run a TAB expansion is part of the keyboard configuration - setup with the first and second keyboard page . Some minor options are under control of other flags.

Here the list of available TAB expansion functions on the second keyboard page - as you see each function comes in two kinds - both use the same settings, but move in opposite directions within the list of found matches.

Expand Path : First function, forwards Expand Backwards : ditto, but backwards Expand Short : Second function, forwards Expand Short Bkwds : ditto, but backwards Expand Devices : Third function, forwards Expand Devs Bkwds : ditto, but backwards Expand Dirs : Fourth function, forwards Expand Dirs Bkwds : ditto, but backwards Expand Icons : Fifth function, forwards Expand Icons Bkwds : ditto, but backwards Expand Alt : Sixth function, forwards Expand Alt Bkwds : ditto, but backwards

The names of the functions are completely arbitrary - you can configure them as you like.

For each of the pairs above, one set of options and priorities is available at the shell pages . The name of the pair currently edited by these pages is printed directly below the headline, and can be selected by the "« Prev" and "Next »" gadgets near the function name.

Here's the list of available options on the third shell page :

Double TAB Requester

First TAB Expands fully Requester if expansion is ambiguous

Add ViNCEd matches to the requester

ViNCEd assigns a priority to each match of the TAB expansion and sorts the list of found matches by priority. The entries of higher priority are then shown earlier, on top of the list; hence, you need less keyboard presses to reach them. Objects with a priority of -128 or below aren't added to the list at all and will be dropped. That means that you may disable certain object classes completely by assigning them a priority of -128.

The priority settings are on the fourth shell page . They are also six sets of priorities, one for each TAB expansion function.

Files Dirs Icons Devices Assigns Volumes Path C: Dir Resident Scripts Executables

Your priorities get "adjusted" a bit if you search for a directory or a device explicitly, i.e. the template ends either with a forwards slash "/" or with a colon ":". In this case, all other file types except directories or devices/assigns/volumes are ignored and the priority of the file type looked for is eventually raised from -128 to -127. This means especially that even if you disabled directories explicitly, searching for an object ending with "/" will actually match something.

There is actually another flag that has some influence on the TAB expansion , and that's the "History buffer wraps around" option on the second shell page . If this option is enabled, the TAB expansion list will "wrap around", too: As soon as one end of the list is reached, ViNCEd will insert a blank line. If now one of the TAB expansion functions is pressed again to move on, the buffer "wraps around" and ViNCEd starts to display the first entry from the other end of the buffer.

Finally, you find on the same page the number of directories which are simultaneously kept in the TAB expansion cache.

1.97 Why does that match commands?

ViNCEd knows that you are searching for a command because you're expanding the first argument on a line, and this argument is neither a directory nor a device - that is, it's name does not end with a ":" or a "/". The consequence is that all locations where commands can be found are considered. That is, the current directory, the "C:" (multi-)assign, all directories in the shell "path", and the list of resident commands.

This can be changed , of course.

1.98 Why does this match commands that start with rex?

ViNCEd extracts the command the cursor is placed in and treads this as a so called template . The rules for these templates are as follows:

All character strings that match the template must begin with the same characters in front of the cursor as in the template, and must end with the same characters as under and behind the cursor in the template. That is, a possible match must "insert" characters at the cursor position - remember that if you would type in characters from the keyboard, these would exactly go between there.

Since the example template starts with "rex", and the cursor is placed behind the "x", all possible matches have to start with "rex". Since no characters are under or behind the cursor, the end of the match is arbitrary.

Another example would be

1.SYS:> lt

which would match all commands that begin with "l" and end with "t".

To express this even in different words for the experts, the Amiga Dos search pattern is build by the template by inserting a "#?" right at the cursor position.

1.99 How to change the command search directory?

Which directories are considered for searching commands is part of the configuration of the TAB expansion keyboard function used for expansion. In our example, the first available keyboard function is relevant since this is the function bound to the Ctrl TAB key.

To change these settings, open the SetVNC program and go to the fourth shell page . The page will already show the settings for the first expansion function, all others can be adjusted by pressing the "« Prev" and "Next »" buttons on this page. In total, six expansion keyboard functions are available.

This page shows now a list of priorities which "weight" all possible matches. As soon as a priority goes below -127, the match is no longer considered.

As far as the example is concerned, the "Resident", "C: Dir" and "Path" gadget is of importance:

The first gadget is the absolute priority for matches in the resident list. Set this to "-128" to ignore resident commands.

The second gadget is a relative priority that is added to matches found in the "C:" directory. If this is set to "-128", this directory will be ignored completely, regardless how high the overall priority may grow.

The third gadget is again a relative priority used for matches in the shell path, i.e. all directories printed by the "Path" command except the current directory and the C: assign. Again, if this is set to "-128", the path will be ignored.

More details about how the priorities are used are described in a different section .

1.100 How to exclude directories?

You can of course setup whether directories should be included in this situation. It's part of the configuration of the TAB expansion keyboard function used whether directories should match or not.

In our example, the first available TAB expansion, which is bound to Ctrl TAB, has been used.

To change its settings, open the preferences editor , the SetVNC program and go to the fourth shell page . The white text below the headline will show now the TAB expansion function to be adjusted, to be controlled with the "« Prev" and "Next »" buttons right to it. The gadgets below setup the priorities assigned to certain objects. You may change here the entry of the "Dirs:" gadget in the first row to a lower value, or probably even to -128; objects with a total priority lower than -127 aren't considered as a valid match, hence this would exclude directories completely.

More details about how the priorities are used are described in a different section .

1.101 How to exclude assignments?

It's part of the ViNCEd configuration which sources are searched to find a match for a given template , or to be precise part of the configurations of one of the six the keyboard functions used for the TAB expansion. In our example, the first available TAB function is relevant since this is the function bound to the Ctrl TAB key used in the tutorial .

To change these settings, open the SetVNC program and go to the fourth shell page . The page will already show the settings for the first expansion function, all others can be adjusted by pressing the "« Prev" and "Next »" buttons on this page. In total, six expansion keyboard functions are available.

This page shows now a list of priorities which "weight" all possible matches. As soon as a priority goes below -127, the match is no longer considered.

The priority that is relevant for the assigns is, obviously, the number in the "Assigns" gadget. Set this to -128 to exclude assignments. Similarly, devices and volume names can be removed as well - by setting their priority to -128 as well.

More details about how the priorities are used are described in a different section .

1.102 How to exclude icon files?

Which files types match a TAB expansion is under control of the ViNCEd configuration, or to be specific, under the configuration of one of the six TAB expansion functions available. In our example, this is the first function available because that is the function that is bound to the Ctrl TAB key.

To change its settings, load the SetVNC program and go to the fourth shell page . This will show the priority settings for the first TAB function already.

To exclude icon files, enter a value of "-128" in the top right "Icons:" gadget on that page - the TAB expansion will ignore all objects with a priority lower than -127.

More details about how the priorities are used are described in a different section .

1.103 Side mark: Inserting spaces

As an exception, ViNCEd will NOT insert spaces after a successful and unique expansion if the match was either a directory or an assign. The last letter will be a forwards slash "/" or a colon ":". This is mainly for convenience in all-day use.

1.104 How to avoid the refinement?

In case you don't want ViNCEd to insert a refined template if more than one possible match was found, you have to adjust the configuration of the TAB expansion keyboard function used. This would be the first available function in our tutorial which is bound to Ctrl TAB by default.

To adjust the settings, load the SetVNC program and go to the third shell page . It will already show the flags of the first TAB expansion function, the others are available by the arrow gadgets "« Prev" and "Next »". In this page, turn on the gadget "First TAB expands fully".

More details about how the flags settings are described in a different section .

1.105 What about a requester if not unique?

If you like to see a requester if more than one possible match of your template was found, you have to adjust the configuration of the TAB expansion keyboard function used. This would be the first available function in the above tutorial which is bound to Ctrl TAB by default.

To change the settings, load the SetVNC program and go to the third shell page . It will already show the flags of the first TAB expansion function, the others are available by the arrow gadgets "« Prev" and "Next »". Turn on the gadget "Requester if expansion is ambiguous" on this page.

More details about how the flags settings are described in a different section .

1.106 About info files and requesters

There are two (or three, to be precise) popular requester packages available - the standard "asl" package which came with your workbench and the "reqtools" replacement which should be patched in by "reqchange" (I don't recommend RTPatch, though). The "asl" library shows the "info" files by default, the reqtools patch does not. However, there should be a tiny ".info" button in each reqtools gadget which allows to include these files as well. The problem is here that all matches found by ViNCEd are again filtered by the requester package. A sometimes useful, but sometimes annoying side effect.

(BTW, the third package is the arp.library which is used if ViNCEd is run under Os 1.2/1.3. The 3.60 does still support these obsolete versions, but support will be dropped in 3.70.)

1.107 How to use standard requesters?

ViNCEd uses a lot of trickery to allow you selecting ALL found matches of a TAB expansion from a file the requester, and even to enter directories and assigns found in this way. If a file was found twice - because it happens to be in two directories ViNCEd looked for expansions in - then it will appear twice in this requester. While this might look weird, it is supposed to be a warning sign to you that it is not clear which version is actually used by the shell.

If you don't like all this magic, you can switch back to a plain file requester. This will, however, ONLY show the files in the current directory level, NO assigns, NO devices, no extras.

The corresponding flag is located on the third shell page of the SetVNC program - one flag is available for each of the six TAB expansion functions. To select the function to modify, use the "« Prev" and "Next »" buttons on top of the page.

1.108 How to turn off the Double-Tab requester?

The Double-Tab requester is invoked by ViNCEd as soon as another TAB expansion keyboard function is used within the double-click period after the TAB expansion has been completed. If this requester is "in the way" because you prefer keyboard based working to a graphical interface (I'm faster in typing than in clicking anyhow...), you may disable this requester function.

To do so, open the SetVNC preferences editor, go to the third shell page and select with the "« Prev" and "Next »" gadgets near the top of the page the TAB function you'd like to adjust. The first flag on this page - "Double TAB requester" - must be un-checked to disable the requester.

1.109 What is a refinement?

A "refinement" of a template is a template that matches exactly the same entries as the original template, but contains more characters. Hence, it is the largest possible "partial expansion" of the template given in first place.

ViNCEd will place the cursor within the "refinement" in such a way that again a valid template is constructed. To remove the ambiguity of this template, or at least to exclude some of the matches, characters have to be inserted right at the cursor position.

1.110 What is a template?

A "template" is a sequence of characters, the cursor placed within this sequence, which is used as a "wild card" to search an entry either of the ViNCEd command history or of the directories for a TAB expansion . Each object - either an entry of the history or a candidate for a TAB expansion is compared against this template and either considered as a match, or ignored.

The rules for matching are quite simple: A candidate must begin with the characters in the template up to the character in front of the cursor position, and must end with the characters under and behind the cursor position. Hence, a template like

```
1.SYS:> lst
```

would match candidates which start with an "l" and end with "st".

To give an intuitive "rule of thumb" for templates: A possible candidate is only allowed to insert characters at the cursor position. It is not allowed to change any other character already given in the template. This means, especially, that you may always refine a template by typing some characters at the cursor position.

1.111 The magic Ctrl Z key in the Shell mode

This is a unique feature of ViNCEd which is mostly known from the Unix world; it is only available if the Shell mode is enabled, and it requires at least Os 3.0 to make it working, or the "NamedConsoleHandler". This handler should be installed as part of the installer script, and should be mounted in the startup-sequence. It is not required for Os 3.0 or later.

What does this, now? It happened more than once that I was running a program in the shell and then decided that I actually need to run another one in between. However, since I forgot to use "RUN" to start the first program, the shell in the window was no longer available as it was "blocked" by the first program already. What I had to do in this situation was to go to the workbench and open another shell; Unix, however, offers a special key just for that situation: Hit Ctrl z to suspend the currently running program and make the shell re-available again.

ViNCEd offers "almost" the same feature for the Amiga: If a program is blocking the shell, hit Ctrl z to launch just another shell in the same window. Output of the first program going into the window while the second shell is working will be "suspended". ViNCEd will not "mix" both outputs and confuse you; only one of the running shells will be allowed to print in the window. The additional scripts "fg" and "bg" implement "almost Unix" "job control" commands. Using these commands, you're able to control which program is "in foreground" and which are not. Only foreground commands are allowed to print on the screen; and if a background program tries so, it will be "suspended", i.e. stopped until you put it "in foreground" explicitly.

How about an example? Open a ViNCEd shell - keep care that it is in Shell mode or this magic won't work. Then enter the command

1.SYS:> listReturn

and while the output is in progress, hit Ctrl z. What you'll see looks approximately like the following:

Disk&Debug Dir ---arwed 14-Jun-98 19:19:03 Disk&Debug.info 628 ---arw-d 10-Jan-97 22:00:27 Disk.info 376 ---arw-d 10-Sep-96 22:16:38

New Shell process 2 2.SYS:> "list" suspended. [ViNCEd output] 2.SYS:>

At the time you pressed Ctrl z, ViNCEd launched a new shell for you, here indicated by the text "New Shell process 2". The prompt of this new shell is already printed on the line below. However, the "list" command was still running and tried to print some output. Since the new shell is now "in foreground", the "list" command has been put in background. As soon as it tries to print something, it is stopped and ViNCEd prints a message for you that this happened in the next line:

"list" suspended. [ViNCEd output]

The prompt of the new shell is then requested again. As soon as you want to see the "list" output, you've to put that "to foreground" again. To do that, you need the "CLI number" of the "list" output; type "status" to see which commands are currently running:

1.SYS:> status Process 1: Loaded as command: list Process 2: Loaded as command: status 10.SYS:>

In our tiny example, the "CLI number 1" is used by the "list" command, and the shell number 2 is running the "status" command you entered. To put now the "CLI 1" to foreground, type

2.SYS:> fg 1 Expansion Dir ---arwed 30-Apr-98 23:46:36 Expansion.info 632 ---arw-d 04-Sep-96 23:34:48

and the "list" command will resume. Looking at the prompt some lines below reveals that you're now back in the "CLI 1" and CLI 2 is in background. However, since nothing is printed by this shell and it's just sitting there and waiting for your input, it won't get suspended. Of course, you may still put it back in foreground with "fg 2":

1.SYS:> fg 2 2.SYS:>

Somewhat more insight for the advanced user about the job control is in a separate section .

1.112 The command history

The command history is another ViNCEd buffer . Unlike the screen "review buffers", it's not directly visible on the screen, but is nevertheless useful. (However, it can be made visible with the history Shell script).

As soon as you entered a command and press Return (or precisely, the Send Inputs keyboard function which is usually bound to this key), your input enters the command history. With the ViNCEd history functions, this input can be restored later on - which helps typing longer command lines over and over again. The difference between the history and the review buffer - which can be used, too, to rerun a command - is that the cursor doesn't leave the current input line at all.

Besides that, there are keyboard functions to "move" in the command history - very much like moving in the display buffer. Even better, you may also search for commands in this buffer.

The following is a tutorial about how to work with the history. The list history relevant settings is in a different section

To demonstrate a bit how this works, enter the following commands a the ViNCEd shell:

1.SYS:> list

and press Return to list the current directory. Next, run

1.SYS:> dir

and, for a last time, enter

1.SYS:> list SYS:

The default configuration binds the functions traversing in the history to the Alt Cursor Up and Alt Cursor Down keys - so let's try these; press Alt Cursor Up:

1.SYS:> list SYS:

This shows again the last command entered. Pressing this key again reveals

1.SYS:> dir

the command entered previous to that - you just moved "two lines" in the history "upwards". Pressing a third time Alt Cursor Up shows of course

1.SYS:> list

and pressing it a fourth time shows just a blank line

1.SYS:>

because you've just encountered the end of the history. Pressing Alt Cursor Up again reveals nothing new, just again the blank line. You may ask ViNCEd to "wrap" the history "around" in this case, i.e. to start from the opposite beginning as soon as you passed the end - instead of just stopping at the end. This is controlled by the History buffer wraps around flag on the second shell page of the SetVNC preferences editor, or alternatively by the "Wrap Around Buffer" item in the Settings menu .

Try now the backwards moving functions: Alt Cursor Down. As expected:

1.SYS:> list

Erase now this command and enter a new one:

1.SYS:> dir SYS:

This shows, again, the directory. But it will, too, "rewind" the history and append this command at the end of it. If you had entered the "list SYS:" command again, ViNCEd would have noticed that there is already another "list SYS:" at the end of the history and won't keep this duplicate entry - even though the history gets rewinded. As always, there's a flag in the SetVNC program which disables this feature and hence allows keeping duplicates; it's located on the second shell page .

Anyways, pressing Alt Cursor Up now shows the command you just entered

1.SYS:> dir SYS:

again, and using the history movement function again shows

1.SYS:> list SYS:

the previous one.

Erase now this line, and press Return on the blank line. This will also "rewind" the history - but the blank line won't enter the history. Instead of moving now "upwards" in the history, you may also start looking at it "downwards", i.e. scan it in "reverse order". Just press now Alt Cursor Down, and ViNCEd prompts with

1.SYS:> list

the very first command entered. Thus, if the history is "rewinded", you're in some sense both at its beginning and its end at the same time. The first movement selects from which end to start - this might come useful as soon as you want to look for a command you entered ages ago.

Since erasing the input line and pressing Return each time you want to rewind the history might be a bit unconvenient, there's a short-cut for this function. It's by default bound to the Ctrl b key; this keyboard function is obviously called Rewind History .

By the way, ViNCEd does not keep an infinite amount of lines in the history, it's by default 128 lines big. The size of the history can be setup on the first window page of SetVNC or by using the Settings menu attached to the window.

Let's come to another useful feature of the history - you may search for commands. For first, rewind the history with Ctrl b. Then enter the following template

1.SYS:> li

and press Amiga B or, alternatively, Shift Alt Cursor Up. Both functions search in the history in the "backwards" or "upwards" direction for a line matching the template - in this case for commands starting with "li".

The rules how templates for searching are build are described in detail elsewhere , but to give an intuitive rule for how they work: ViNCEd scans the history buffer and looks for lines that are formed just by inserting characters at the cursor position. Other modifications are not allowed. Hence, a template like

1.SYS:> lt

would allow ViNCEd to insert characters between the "l" and the "t", but nothing else. Hence, lines starting with "l" and ending with "t" would match here.

Going back to the tutorial, ViNCEd finds the following line:

1.SYS:> list SYS:

the last available command that starts with "li". To search on, press Shift Alt Cursor Up again or Amiga B, whatever you like. ViNCEd remembers the last pattern and will resume scanning the history from the last found match. Hence, you'll see the following line on the screen:

1.SYS:> list

which is the very first command you entered. Searching on reveals no other matches, it will just flash the screen, obviously (try it!).

If you're left at a certain point in the history, you may of course search in the opposite direction as well, towards the "newer" entries or the "bottom end" of it. This works either with Amiga F or Shift Alt Cursor Down.

This ends the tutorial session - the list of flags that control the history function is listed in a different section .

1.113 All history settings at once

The history is mainly controlled by two flags which are found on the second shell page of SetVNC . The first one,

History buffer wraps around

controls what happens if the end of the history is reached while moving in the history. If this flag is disabled, the history movement functions will just stop and present a blank line. If the flag is enabled, the history will "wrap around" and show again the first line of the opposite end of the history buffer. This flag controls, too, the Tab expansion list.

The second flag,

Keep duplicates in the history

is used to tell ViNCEd what to with "repeated commands". If the command from the "bottom" or "newest" end of the history is entered again, this command will enter the history only if this flag is checked. If the flag is not checked, ViNCEd will rewind the history, as usual, but will simply ignore the line as far as the history is concerned.

The size of the history, in lines, is under control of the first window page . It defaults to 128 lines. If the size of the history grows larger than this, the oldest line is thrown away to make room for the latest one.

Finally, the keys used for the history are setup on the keyboard pages . The following keyboard functions are relevant:

History Up : Moves in the history, towards older entries History Down : Moves in the history, towards newer entries Search Partial Upwards : Interprets the current input line UP TO the cursor position as template and searches for this template in the history towards older entries. This function is provided for backwards compatibility with the CON: mode of history searching which does not use the ViNCEd template mechanism. Search Partial Downwards : Similarly, but in downwards direction. Search History Upwards : Interprets the complete current input line as template and searches for this template in the history towards the older lines, as described by the tutorial . Search History Downwards : Ditto, but searches towards newer lines.

1.114 The scripts contained in this package

Seven scripts have been copied by the installation disk to your S: drawer Three of them are part of the job control , and the others replace standard tools usually found in the C: drawer.

S:fg

is used for job control and calls SetVNC to put a specified CLI to foreground . The argument is the number of the CLI process that should be put to foreground, as printed by the "status" command.

S:bg

another job control script which sends the current shell to the background using SetVNC . It does not take any arguments.

S:fork

starts a new shell in the current shell window and sends it to foreground. If an argument is supplied, this command with all remaining arguments is executed in background. Mostly provided as an example how to use named consoles and job control .

REMARK: There's one quirk about the "fork" command: If you run a program with it, its input and output are set to the new "background" owner, but its controlling terminal, which is used to open a "*" file, will remain the current stream. Thus opening "*" from such a process *MIGHT* end in a deadlock situation! This is due to a gap in the RUN command, and some others in the way how the AmigaDOS executes script files. There's currently no way to fix this with just a script, so take this as a demonstration what is possible. A replacement "Run" should be able to fix this

S:SetKeyboard

replaces the C:SetKeyboard command and selects the keyboard for ViNCEd windows. The old C:SetKeyboard still works, but it is more a "hack" than a stable implementation. The script uses the recommended way thru documented Esc sequences. The needed argument is the name of the keyboard, e.g.

1.SYS:> SetKeyboard dReturn

selects the german keyboard.

S:SetFont

replaces C:SetFont and selects the terminal font. The old C:SetFont still works, but is like C:SetKeyboard more a hack and conceptionally not clean. The supplied script uses the documented Esc sequences to do the same job much better and safer. However, this script does not check for proportional fonts and uses them in the window without warning. As a result, the display might look very ugly and the input of commands will be confusing with proportional fonts, moreover, some graphical artifacts may show up. Except that, the arguments are identical to C:SetFont. Maybe somebody wants to write a replacement for C:SetFont that takes care of ViNCEd? In that case, contact me .

S:More

is a workaround for the CBM "More" page utility. It is assumed that you keep the original More program in the drawer "SYS:Utilities". If your "more" is located somewhere else, please edit the script!

The script frees the window pointer after "More" quit, in order to re-establish the iconification .

S:History

dumps the history to the screen or to a file and demonstrates the power of the "PUT" and "GET" arguments of SetVNC .

1.115 Details about Job Control

A new feature introduced with ViNCEd is the so-called "job control". A "job" is just couple of ordinary shell commands that share an common meaning of "foreground" or "background".

There is only one "foreground" job - which is allowed to receive inputs and to print on the screen, and arbitrary many "background" jobs. The commands running as part of these jobs neither receive input, nor may print anything to the screen. In case they try, they just get "suspended", i.e. halted until you bring them to "foreground" manually.

An important part of the "job control" managed by ViNCEd is the Ctrl z function. It aborts the current foreground process, brings it to background and launches a new shell in foreground. A tutorial about this key can be found in the Ctrl z section.

Three scripts for "job control" have been installed in your "S:" directory as part of the installation process. Let's check how to use them:

Open a new shell if not already done. Since you should be able to run commands as usual in this shell, the "job" related to this shell is in "foreground". We're now starting a second shell in the same window and put the first shell to background:

1.SYS:> forkReturn

This will print some messy lines on the screen - which are due to the way how the script works - and ends up with a shell prompt of a different process number:

3.SYS:>

The "CLI number one" is now in "background", the "CLI three" is in foreground and ready for input. We may now explicitly put the new shell to background and bring the "Cli one" to foreground again. This works with

3.SYS:> bgReturn

the "bg" - short for "background" - command. Since just moving "shells" back and forth doesn't help too much, let's try something useful. Run the "list" command in the first shell

1.SYS:> listReturn

and "suspend" it in the middle of the output by pressing Ctrl z. It will be aborted, put into background and the next available shell will be brought to foreground:

Devs Dir ---arwed 10-Jul-98 00:10:53 Devs.info 632 ---arw-d 05-Sep-96 21:22:27 3.SYS:> "list" suspended. [ViNCEd output] 3.SYS:>

At first the shell "Cli 3" shows up, and "list" is put to background. However, since "list" still tries to continue its output, it will get suspended, i.e. halted. The "Cli 3" prompt is now printed again and ready for input.

To resume the "list" program, you've to put it back to foreground. That happens either implicitly with the "bg" command by putting the currently running shell "back", or by using the "fg" command and a "Cli number" explicitly. Since "list" runs in "Cli 1" in our tutorial, let's try:

3.SYS:> fg 1Return

and the output of "list" resumes. "Cli 3" is now in background again.

By the way: The ViNCEd notion of "foreground" and "background" is not related to the AmigaDos notation of "background processes". A program launched with "Run" is, in AmigaDos terminology a "background process". However, it is still allowed to print on the screen since it shares the "job" with the shell it was started with.

Hence,

1.SYS:> run listReturn

will make no difference from what you're used to. However, you won't be able to abort this output by Ctrl z since this program is no longer run as a part of the current shell. A better and job control compliant way to run "real background processes" is by using the "fork" command together with an argument. For example,

1.SYS:> fork listReturn

will not only run the "list" command, it will also put it to background. Hence, the next line of output will be

"list" suspended. [ViNCEd output] [CLI 2] 1.SYS:>

to indicate that "list" was suspended because it tried to print on the screen. Commands that do not try to print will be of course allowed to run on as usual.

To go on, let's put "list" back in foreground again. Do to so, we need first it's "Cli number". This is printed by the status command:

1.SYS:> status Process 1: Loaded as command: status Process 2: Loaded as command: list Process 3: No command loaded

In this example, it's the "Cli 2" that runs "list". Hence, to resume output, enter

1.SYS:> fg 2Return

and the listing will go on. You will, however, still end up in the "Cli 1"

1.SYS:>

since the "job" that was "forked" to run "list" quits as soon as the command quits. In this case, ViNCEd will select the next available shell for you and bring it to foreground.

Some additional nodes about the "job control scripts": They use all the SetVNC program for its function, namely "SetVNC foreground" and "SetVNC background". ViNCEd - or SetVNC to be precise - will NOT allow you to control jobs that have been launched from a different ViNCEd window than the current one - mainly because of security reasons.

However, there's one tiny exception: It may happen that you somehow "got stuck" because Ctrl z doesn't seem to work anymore and you won't be able to bring a shell back to foreground, even if there's one available in that window. This happens for example if ViNCEd can't abort a running process safely since there's currently not enough information about it available. If this happens, you've to open a new shell window, unfortunately. You *may try* to bring the shell running in the other shell window to foreground with a "brute force" method:

First, find its "Cli number" by running the "status" command. The lines showing

Process 2: No command loaded

are in principle available, even though "status" does NOT say in which window these shells are actually launched. Anyways, to bring this shell to foreground, even though it is not running in the same window than the window SetVNC will be run from, enter

1.SYS:> SetVNC foreground other 2

The "other" keyword will tell SetVNC to ignore that the "Cli 2" is actually running in a different window and it will try to bring that to foreground anyways - usually successful.

However, BE WARNED! This operation is not completely safe. It might crash your computer in certain, very delicate situations.

Probably a last word for the experts: The ViNCEd "jobs" are implemented thru so called named consoles - which are part of the ViNCEd console owner system - which is another ViNCEd speciality. Named consoles are only supported by Os versions 3.0 and 3.1 - which is the reason why this doesn't work for Os 2.1 or below.

1.116 Compatibility notes

More

Just a couple of bugs in this program. First, it requests a pointer to the intuition window without giving it back. This is common to all "CON:" programs and will disable iconification and closing of AUTO windows. As a workaround, use the "More" script , which is part of the ViNCEd distribution. It will free the window pointer afterwards for you. Read also about the FreePointer command line option of SetVNC .

Second, "more" expects "Close-Window" events without asking for them. I added a fix for that: Whenever a window is switched to raw mode, the close window event is turned on. Sigh.

A fix for the iconification bug is now available for the 40.3 version of "More" and can be installed optionally.

Ed

Again, the same pointer problem as above. I said this is usual, but not my trouble. Second, "Ed" sends the undocumented CSI sequence "CSI 1K" which should, according to the VT-220 standard, erase the beginning of a line. Instead, "Ed" expects ViNCEd to erase the end of the line. Added a workaround for this, "CSI 1K" is now interpreted in the wrong way in CBM compatibility mode, but works like it should in VT-220 mode. Third, "Ed" leaves a pending Read packet in the stream of ViNCEd. This packet is now canceled each time a Close Event is send to the owner used by "Ed", together with all other packets pending there. Not very nice, in fact, but prevents crashes and I can't think of another solution. Last, to use "Ed" in a ViNCEd window, you should change the scrolling behaviour by some CSI sequences, with "CSI >?18l" "CSI >?19l" and change maybe the block control with "CSI >?25l". Read more about the control codes in the control code section .

pdksh resp. ksh

This unix korn shell implementation is another program that does not give back the window pointer it receives. Unless other programs, it asks for the window pointer EACH TIME it displays a prompt - to find out if it is reading from a console, I guess - making iconification completely impossible, even with a "SetVNC FreePointer". The reason is that after freeing the pointer with this command, it asks again for a new pointer if the prompt gets displayed, ARGHH! The implementation of this function in "ixemul" should be really fixed, since there's currently NO solution for this problem. TAB expansion in "pdksh" windows won't work. The reason is the rather strange reading mechanism of the pksh which somehow emulates the timed/nonblocking reads of unix. Instead of sending a read request to the console, a WaitForChar() style mechanism is used. However, these packets aren't accepted by ViNCEd as stable basis for a TAB expansion, as the packet might get answered at a time not fully under control of ViNCEd - and the CurrentDir() of the waiting process might get lost in between. There's currently no way to fix this. A private TAB expansion like in the unix "bash" would be the better and cleaner solution anyways, so I won't support this bad style.

Z (The Aztec Editor, ancient!)

Well, if you MUST use this editor (maybe you like VI as well... Argh!), here is how: Send "CSI >?18l" "CSI >?19l" to turn off the ViNCEd scrolling. See also the ViNCEd control sequences .

asm (The lattice assembler, old versions 5.xx)

This program has an illegal control sequence in its title. When printed, ViNCEd inserts 1988 blank lines. This might take a while, but is not dangerous. The new versions work well.

csh,ZShell,NewShell

Sends an illegal CSI sequence to ViNCEd ("CSI q" instead of "CSI 0q"). I added a workaround for this, but don't expect this to work any longer. It looks like this "CSI q" sequence is very popular and seems to be documented somewhere. I have no idea who had the strange idea to invent this sequence instead of taking the documented one.

Additional, it does a lot of switching between raw mode and cooked mode, which is quite unnecessary, since the editor features of ViNCEd are superior to that of csh - I advice you to use the "-a" flag. Again, since "csh" reads the window pointer and does not give it back, ViNCEd can't iconify this window anymore (different to KingCON, which still allows the user to iconify the window and, hence, crashes the system). A second problem is, also I haven't checked that out, that the internal commands of "csh" are not in the resident list of the DOS (I suppose). For that reason, they can't get expanded by the TAB key. I advice you to place some dummy commands in the resident list or in the C: directory.

CenterTitles

Not a real bug, but the window title might look a bit messy for one or two seconds after opening a ViNCEd window with buttons in it. This messed window title will go away almost immediately and is harmless (and due to a bug in CenterTitles which uses the wrong draw mode to refresh the title.)

ScreenShell

The output of ViNCEd with that program might look a bit different from how CON: outputs looked like. This is due to the fact that ViNCEd does not ignore the settings of the screen pens, different from CON: which does not care about the screen pens at all.

AFS ("Anti File System")

This is actually not a problem with ViNCEd, but a problem with the ExAll() patch installed by SetPatch 43.6 and later versions, installed for the V39 ROMs. The SetPatch code adds a workaround for a bug in the DirCache filing system which conflicts with AFS due to another bug in AFS. (ARGH!) ViNCEd tries to solve this problem by using its own ExAll() routine for these ROM versions. This is, again, an AFS bug and NOT ViNCEd related.

Vim

As all window editors, it expects window scrolling disabled. Even more, to make the vim build-in block operations work, the insertion of scrolled lines into the lower display buffer should be disabled. Some advanced versions of vim use the extended color highlighting feature, which should be enabled to make use of the extended colors of ViNCEd instead of printing these lines in boldface. The CSI sequences that should be sent as part of the startup process are "CSI >?18l" and "CSI >?19l" to disable scrolling, "CSI >?14h" to enable the extended colors instead of bold and "CSI >?13l" to disable scrolling of lines into the lower display buffer. These CSI sequences should actually be part of the vim "termcap" configuration.

XEN.8 font

The XEN.8 font, even though supposed to be a fixed width font, is in fact a proportional font. The container size of some characters is actually LARGER than the font width - the font is declared to be six pixels wide, but the "f" character has a width of seven pixels. This might cause some graphical artifacts, but is otherwise harmless. Replace it by the topaz6.8 font in the ViNCEd archive.

1.117 Compatibility notes for the experts

Some programs don't follow the compatibility guide lines formulated in the Rom Kernal Reference Manuals . Some of these problems have been fixed by work-arounds I will remove sometimes.

ViNCEd compatibility guidelines

-Do not send any undocumented CSI sequences. If you want to stay compatible with CON:, send only codes documented in the Rom Kernal Reference Manuals .

-If you want to find out whether a window belongs to ViNCEd, use the "FindCNWindow()" function of the library. It also provides a pointer to the intuition window. Call "UnFindCNWindow()" to give this pointer back. If you don't, the window can't be iconified later on.

-Do not expect that the stream is linked to an open intuition window. It might be closed by a iconification request any time. To make sure THAT a valid intuition window exists, call "FindCNWindow()".

-Do not expect your process is in foreground. Printing from a background process might suspend your program. Read the job control section and the console owner section for more details.

-Do not expect that you will receive any raw events you haven't requested, but be prepared that you might even ask for them in the "cooked" mode.

-Do not expect that the handler will break lines somewhere. Use the control sequences to select the mode you need.

-Do not poke in the console unit you received with ACTION_DISK_INFO. Changes in this structure are ignored by ViNCEd. In special, do not set the the font in that structure and do not set the keymap, even though this MIGHT work as a part of a compatibility hack. ViNCEd supports CSI sequences to do that, read control sequence section .

-Do not expect that the startup ID -1 will work. This lesser known feature of the 1.3 console handler was used to attach the console to a previously opened window, but is no longer supported - neither by the newer CON: handlers nor by ViNCEd. Use the WINDOW open path argument, or the ConMan style "W" argument.

1.118 List of all Keyboard Functions

Cursor Left (Keyboard Function)

Cursor Right (Keyboard Function) Cursor Up keyboard (Keyboard Function)

Cursor Down keyboard (Keyboard Function) History Up (Keyboard Function)

History Down (Keyboard Function) Search Partial Upwards (Keyboard Function)

Search Partial Downwards (Keyboard Function) Search History Upwards (Keyboard Function)

Search History Downwards (Keyboard Function) Half Screen Left (Keyboard Function)

Half Screen Right (Keyboard Function) Half Screen Up (Keyboard Function)

Half Screen Down (Keyboard Function) To Left Border (Keyboard Function)

To Right Border (Keyboard Function) To Top of Screen (Keyboard Function)

To Bottom of Screen (Keyboard Function) Prev Word (Keyboard Function)

Next Word (Keyboard Function) Prev Component (Keyboard Function)

Next Component (Keyboard Function) Home (Keyboard Function)

End (Keyboard Function) Scroll Up (Keyboard Function)

Scroll Down (Keyboard Function) Scroll Half Screen Up (Keyboard Function)

Scroll Half Screen Down (Keyboard Function) Send Inputs (Keyboard Function)

Split Line (Keyboard Function) Insert ˆJ (Keyboard Function)

Send Complete Line (Keyboard Function) Line Feed (Keyboard Function)

TAB Forwards (Keyboard Function) TAB Backwards (Keyboard Function)

Expand Path (Keyboard Function) Expand Backwards (Keyboard Function)

Expand Short (Keyboard Function) Expand Short Bkwds (Keyboard Function)

Expand Devices (Keyboard Function) Expand Devs Bkwds (Keyboard Function)

Expand Dirs (Keyboard Function) Expand Dirs Bkwds (Keyboard Function)

Expand Icons (Keyboard Function) Expand Icons Bkwds (Keyboard Function)

Expand Alt (Keyboard Function) Expand Alt Bkwds (Keyboard Function)

Send ˆC (Keyboard Function) Send ˆD (Keyboard Function)

Send ˆE (Keyboard Function) Send ˆF (Keyboard Function)

Send ˆC to All (Keyboard Function) Send ˆD to All (Keyboard Function)

Send ˆE to All (Keyboard Function) Send ˆF to All (Keyboard Function)

Delete Forwards (Keyboard Function) Delete Backwards (Keyboard Function)

Delete Full Line (Keyboard Function) Cut Full Line (Keyboard Function)

Delete Inputs (Keyboard Function) Cut Inputs (Keyboard Function)

Delete Word Fwds (Keyboard Function) Cut Words Fwds (Keyboard Function)

Delete Word Bkwds (Keyboard Function) Cut Word Bkwds (Keyboard Function)

Delete Component Fwds (Keyboard Function) Cut Component Fwds (Keyboard Function)

Delete Component Bkwds (Keyboard Function) Cut Component Bkwds (Keyboard Function)

Delete End of Line (Keyboard Function) Cut End of Line (Keyboard Function)

Delete Start of Line (Keyboard Function) Cut Start of Line (Keyboard Function)

Delete End of Display (Keyboard Function) Form Feed (Keyboard Function)

Clear Screen (Keyboard Function) Cut (Keyboard Function)

Copy (Keyboard Function) Paste (Keyboard Function)

Hide (Keyboard Function) Select All (Keyboard Function)

Copy Quiet (Keyboard Function) Reset (Keyboard Function)

Full Reset (Keyboard Function) Toggle ESC (Keyboard Function)

Toggle NumLock (Keyboard Function) Toggle Overwrite (Keyboard Function)

Suspend (Keyboard Function) Resume (Keyboard Function)

Abort Expansion (Keyboard Function) Scroll to Cursor (Keyboard Function)

Rewind History (Keyboard Function) Yank (Keyboard Function)

Generate EOF (Keyboard Function) Display Beep (Keyboard Function)

Toggle Pause (Keyboard Function) Help (Keyboard Function)

Fork New Shell (Keyboard Function) Insert CSI (Keyboard Function)

Insert ESC (Keyboard Function)

1.119 Cursor Left (Keyboard Function)

Moves the cursor one character to the left.

1.120 Cursor Right (Keyboard Function)

Moves the cursor one character to the right.

1.121 Cursor Up keyboard (Keyboard Function)

Move the cursor one line upwards on the screen. If the cursor reaches the upper end of the buffer , it either inserts a blank line or stops. This depends on the Don’t scroll into the border flag on the first editor page of the SetVNC program .

1.122 Cursor Down keyboard (Keyboard Function)

Move the cursor downwards one line on the screen. If you reach the lower end of the display buffer , ViNCEd inserts a blank line or stops the cursor there. What happens depends on the Don’t scroll into the border flag on the first editor page of the SetVNC program .

1.123 History Up (Keyboard Function)

Inserts the next available older line of the command history at the current cursor position, and moves the history pointer one line towards the older end of the history.

If the end of the history is reached, ViNCEd inserts either a blank line or "wraps around" to the start of the history. This depends on the History buffer wraps around flag on the second shell page of SetVNC .

1.124 History Down (Keyboard Function)

Inserts the next available newer line of the command history at the current cursor position, and moves the history pointer one line towards the newer end of the history.

If the start of the history is reached, ViNCEd inserts either a blank line or "wraps around" to the end of the history. This depends on the History buffer wraps around flag on the second shell page of SetVNC .

1.125 Search Partial Upwards (Keyboard Function)

Takes the user input at the current line up to the cursor position as template to search in the command history starting from the current history pointer in upwards direction, i.e. towards older lines.

1.126 Search Partial Downwards (Keyboard Function)

Takes the user input at the current line up to the cursor position as template to search in the command history starting from the current history pointer in downwards direction, i.e. towards newer lines.

1.127 Search History Upwards (Keyboard Function)

Uses the user input at the cursor position as template to search in the command history starting from the history pointer in upwards direction, i.e. towards older lines.

1.128 Search History Downwards (Keyboard Function)

Uses the user input at the cursor position as template to search in the command history starting from the history pointer in downwards direction, i.e. towards newer lines.

1.129 Half Screen Left (Keyboard Function)

Moves the cursor either to the left edge of the visible area, or scrolls the window leftwards one half window width if it is already at the left edge of the window. The cursor stops moving at the right border of printed output, i.e. usually at the prompt of the shell.

1.130 Half Screen Right (Keyboard Function)

Moves the cursor either to the right edge of the visible area, or scrolls the window rightwards one half window width if it is already at the right edge.

1.131 Half Screen Up (Keyboard Function)

Moves the cursor either to the top edge of the window, or scrolls the text upwards one half window height if the cursor is already at the top edge of the window. If the cursor reaches the upper end of the display buffer , ViNCEd inserts either blank lines or stops there, if the Don’t scroll into the border flag on the first editor page is set.

1.132 Half Screen Down (Keyboard Function)

Moves the cursor either to the bottom edge of the window, or scrolls the text downwards one half window height if the cursor is already at the bottom edge of the window. If the cursor reaches the lower end of the display buffer , ViNCEd inserts either blank lines or stops there, if the Don’t scroll into the border flag on the first editor page is set.

1.133 To Left Border (Keyboard Function)

Moves the cursor leftwards either to the start of the line or at least to the start of the user input data.

1.134 To Right Border (Keyboard Function)

Moves the cursor rightwards to the end of the line.

1.135 To Top of Screen (Keyboard Function)

Moves the cursor to the top of the display buffer .

1.136 To Bottom of Screen (Keyboard Function)

Moves the cursor to the bottom of the display buffer .

1.137 Prev Word (Keyboard Function)

Moves the cursor to the start of the current word or, if the cursor is already at the start of a word, move it to the start of the previous word.

1.138 Next Word (Keyboard Function)

Moves the cursor to the start of the next word .

1.139 Prev Component (Keyboard Function)

If ViNCEd is in shell mode , this keyboard function moves the cursor to the start of the current component . If the cursor was already placed there, it gets moved to the start of the previous component .

If ViNCEd is NOT in shell mode, this works identical to the Prev Word keyboard function .

1.140 Next Component (Keyboard Function)

If ViNCEd is in shell mode , this keyboard function moves the cursor to the start of the next component .

If ViNCEd is NOT in shell mode, this works identical to the Next Word keyboard function .

1.141 Home (Keyboard Function)

Moves the cursor to the leftmost position of the topmost row of the display buffer .

1.142 End (Keyboard Function)

Moves the cursor to the end of the bottommost row of the display buffer .

1.143 Scroll Up (Keyboard Function)

Scrolls the contents of the window downwards one line, hence moves the buffer position upwards. The cursor position is "not changed"; this means however different things, dependent on whether the XTerm/CON: cursor mode is set or not. If this flag is not set, the cursor will keep its physical position on the screen and the buffer will scroll under the cursor; that is, the cursor position relative to the text WILL change, even though the cursor does not move. If the flag IS set, the cursor will move with the buffer, and hence move physical. However, it will not change its position relative to the buffer.

This flag can be found on the first edit page of the SetVNC program .

1.144 Scroll Down (Keyboard Function)

Scrolls the contents of the window upwards one line, hence moves the buffer position downwards. The cursor position is "not changed"; this means however different things, dependent on whether the XTerm/CON: cursor mode is set or not. If this flag is not set, the cursor will keep its physical position on the screen and the buffer will scroll under the cursor; that is, the cursor position relative to the text WILL change, even though the cursor does not move. If the flag IS set, the cursor will move with the buffer, and hence move physical. However, it will not change its position relative to the buffer.

This flag can be found on the first edit page of the SetVNC program .

1.145 Scroll Half Screen Up (Keyboard Function)

Scrolls the contents of the window downwards half the window-size downwards, hence moves the buffer position upwards. The cursor position is "not changed"; this means however different things, dependent on whether the XTerm/CON: cursor mode is set or not. If this flag is not set, the cursor will keep its physical position on the screen and the buffer will scroll under the cursor; that is, the cursor position relative to the text WILL change, even though the cursor does not move. If the flag IS set, the cursor will move with the buffer, and hence move physical. However, it will not change its position relative to the buffer.

This flag can be found on the first edit page of the SetVNC program .

1.146 Scroll Half Screen Down (Keyboard Function)

Scrolls the contents of the window downwards half the window-size upwards, hence moves the buffer position downwards. The cursor position is "not changed"; this means however different things, dependent on whether the XTerm/CON: cursor mode is set or not. If this flag is not set, the cursor will keep its physical position on the screen and the buffer will scroll under the cursor; that is, the cursor position relative to the text WILL change, even though the cursor does not move. If the flag IS set, the cursor will move with the buffer, and hence move physical. However, it will not change its position relative to the buffer.

This flag can be found on the first edit page of the SetVNC program .

1.147 Send Inputs (Keyboard Function)

Collects the user inputs from the current cursor line and sends them as to whatever program that wants to receive them, i.e. which called Read() on a ViNCEd stream. Printed characters as the shell prompt and other output will be ignored and is not being sent. The user inputs are then appended to the command history . A new blank line is inserted below the current line, and the cursor is placed at the beginning of the inserted line.

Hence, this implements the usual "Return" key function for a shell.

If the Standard CR insertion at start of line is not checked, this function behaves slightly different at the start of the line. Instead of inserting a line below the current line, a blank line is inserted ON TOP of the current line. The cursor is then moved to the lower, old line.

1.148 Split Line (Keyboard Function)

Inserts a new line right under the current line and splits the current line at the cursor position. The characters behind and under the cursor enter the inserted line, the characters in front of the cursor remain at their position. The cursor is then set to the beginning of the inserted line. At no point any data is sent to a program, nor is the command history altered in any way.

If the Standard CR insertion at start of line is not checked, this function behaves slightly different at the start of the line. Instead of inserting a line below the current line, a blank line is inserted ON TOP of the current line. The cursor is then moved to the lower, old line.

Hence, this implements the "Return" key function of an editor.

1.149 Insert ˆJ (Keyboard Function)

Inserts a Ctrl-J = LF ASCII character at the cursor position. This character will appear as an inverse J on the screen.

Using this command, two or more AmigaDos commands can be put in one line and will be executed sequentially. For example, the command line

1.SYS:> listJdir

will list the current directory twice, first with the "list" command and a second time with the "dir" command.

1.150 Send Complete Line (Keyboard Function)

Collects the all inputs from the current cursor line and sends them as to whatever program that wants to receive them, i.e. which called Read() on a ViNCEd stream. This keyboard function will sent all characters on the line, including printed characters, and will append the complete line "as is" to the command history .

In order not to make this completely useless by sending the shell prompt, this function implements a special "trick": If the cursor is placed on the same line, behind the position where the last output left it, then only the characters starting from that position up to the end of the line are sent. If the cursor is moved in front of this position, or to a different line, then ALL characters, including a possible shell prompt are sent.

This method was already used by the Atari XL operating system and worked well in the 8-bit times, so this will hopefully prove useful as well.

1.151 Line Feed (Keyboard Function)

Inserts a blank line under the current line and moves the cursor to the start of the blank line. Does not sent any data, it just moves the cursor, nothing more.

1.152 TAB Forwards (Keyboard Function)

In insertion mode , this function inserts blank spaces up to the next tabulator stop, and places the cursor at this tabulator stop.

No spaces are inserted in overwrite mode , only the cursor gets moved.

1.153 TAB Backwards (Keyboard Function)

In insertion mode , this function removes characters backwards, starting at the cursor position until the previous tabulator stop is reached. The cursor is then moved to that position.

Nothing is removed in overwrite mode , only the cursor gets moved.

1.154 Expand Path (Keyboard Function)

The first TAB expansion function in forwards moving direction.

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.155 Expand Backwards (Keyboard Function)

The first TAB expansion function in backwards moving direction.

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.156 Expand Short (Keyboard Function)

The second TAB expansion function in forwards moving direction.

The name of this function indicates that it should be used to run a TAB expansion using a shorter search path, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.157 Expand Short Bkwds (Keyboard Function)

The second TAB expansion function in backwards moving direction.

The name of this function indicates that it should be used to run a TAB expansion using a shorter search path, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.158 Expand Devices (Keyboard Function)

The third TAB expansion function in forwards moving direction.

The name of this function indicates that it should be used to expand a device name, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.159 Expand Devs Bkwds (Keyboard Function)

The third TAB expansion function in backwards moving direction.

The name of this function indicates that it should be used to expand a device name, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.160 Expand Dirs (Keyboard Function)

The fourth TAB expansion function in forwards moving direction.

The name of this function indicates that it should be used to expand a directory name, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.161 Expand Dirs Bkwds (Keyboard Function)

The fourth TAB expansion function in backwards moving direction.

The name of this function indicates that it should be used to expand a directory name, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.162 Expand Icons (Keyboard Function)

The fifth TAB expansion function in forwards moving direction.

The name of this function indicates that it should be used to find icon names, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.163 Expand Icons Bkwds (Keyboard Function)

The fifth TAB expansion function in backwards moving direction.

The name of this function indicates that it should be used to find icon names, and the default settings are setup just in this way. However, you may use this function for whatever you like simply by changing its settings .

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.164 Expand Alt (Keyboard Function)

The sixth TAB expansion function in forwards moving direction.

This alternate expansion function is reserved for whatever you might need it for - just adjust the settings to whatever pleases you.

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.165 Expand Alt Bkwds (Keyboard Function)

The sixth TAB expansion function in backwards moving direction.

This alternate expansion function is reserved for whatever you might need it for - just adjust the settings to whatever pleases you.

Reads the argument at the cursor position, interprets it as a template and searches for possible expansions. Details about this process are found in a different section .

1.166 Send ˆC (Keyboard Function)

Sends the break signal 12 to the currently running process. The signal gets NOT transmitted to background processes, such as processes started with "Run" or processes launched with the "fork" script .

This signal is mainly used to abort a running program.

1.167 Send ˆD (Keyboard Function)

Sends the break signal 13 to the currently running process. The signal gets NOT transmitted to background processes, such as processes started with "Run" or processes launched with the "fork" script .

This signal is mainly used to abort shell scripts.

1.168 Send ˆE (Keyboard Function)

Sends the break signal 14 to the currently running process. The signal gets NOT transmitted to background processes, such as processes started with "Run" or processes launched with the "fork" script .

This signal is not widely used in the current AmigaDos.

1.169 Send ˆF (Keyboard Function)

Sends the break signal 15 to the currently running process. The signal gets NOT transmitted to background processes, such as processes started with "Run" or processes launched with the "fork" script .

This signal is not widely used in the current AmigaDos.

1.170 Send ˆC to All (Keyboard Function)

Sends the break signal 12 to the currently running process and all processes started with "Run" that share the same owner as the foreground process. These are usually called "background processes" in the AmigaDos terminology - however, "background" means something different for the ViNCEd job control mechanism. These "real background" processes are not affected by this function.

This signal is mainly used to abort a running program.

1.171 Send ˆD to All (Keyboard Function)

Sends the break signal 13 to the currently running process and all processes started with "Run" that share the same owner as the foreground process. These are usually called "background processes" in the AmigaDos terminology - however, "background" means something different for the ViNCEd job control mechanism. These "real background" processes are not affected by this function.

This signal is mainly used to abort shell scripts.

1.172 Send ˆE to All (Keyboard Function)

Sends the break signal 14 to the currently running process and all processes started with "Run" that share the same owner as the foreground process. These are usually called "background processes" in the AmigaDos terminology - however, "background" means something different for the ViNCEd job control mechanism. These "real background" processes are not affected by this function.

This signal is not widely used in the current AmigaDos.

1.173 Send ˆF to All (Keyboard Function)

Sends the break signal 15 to the currently running process and all processes started with "Run" that share the same owner as the foreground process. These are usually called "background processes" in the AmigaDos terminology - however, "background" means something different for the ViNCEd job control mechanism. These "real background" processes are not affected by this function.

This signal is not widely used in the current AmigaDos.

1.174 Delete Forwards (Keyboard Function)

Deletes the character in front of the cursor and scrolls the remaining characters to the left. Thus, this implements the standard function of the Del key.

1.175 Delete Backwards (Keyboard Function)

Deletes the character in front of the cursor and scrolls the rest of the line to the left. Thus, this implements the standard function of the Backspace key.

1.176 Delete Full Line (Keyboard Function)

Removes the current cursor line completely and scrolls the lines below upwards.

1.177 Cut Full Line (Keyboard Function)

Removes the current cursor line completely and places all user inputs of that line in the yank buffer . The lines below the deleted line are scrolled upwards.

1.178 Delete Inputs (Keyboard Function)

Removes all user inputs from the current cursor line.

1.179 Cut Inputs (Keyboard Function)

Removes all user inputs from the current line and places them in the yank buffer .

1.180 Delete Word Fwds (Keyboard Function)

Deletes all inputs right to and under the cursor in the current word , up to the end of the word and scrolls the remaining characters inwards.

1.181 Cut Words Fwds (Keyboard Function)

Deletes all inputs under and right to the cursor up to the end of the word the cursor is placed in. The deleted characters are copied into the Yank buffer and the remaining characters are scrolled leftwards.

1.182 Delete Word Bkwds (Keyboard Function)

Deletes all inputs left of the cursor up to the beginning of the word under the cursor, and scrolls the remaining characters backwards.

1.183 Cut Word Bkwds (Keyboard Function)

Deletes all inputs left to the cursor up to the beginning of the word the cursor is placed in, copies these characters to the Yank buffers and scrolls the remaining characters inwards.

1.184 Delete Component Fwds (Keyboard Function)

Deletes all inputs under and right to the cursor up to the end of the component and scrolls the remaining characters inwards.

1.185 Cut Component Fwds (Keyboard Function)

Removes all inputs right to and under the current cursor position up to the end of the component , copies the characters to the Yank buffer and scrolls the remaining line inwards.

1.186 Delete Component Bkwds (Keyboard Function)

Deletes all inputs left to the cursor position up to the beginning of the component and scrolls the rest of the line leftwards.

1.187 Cut Component Bkwds (Keyboard Function)

Removes all inputs left to the cursor position up to the beginning of the component under the cursor, copies the deleted characters to the Yank buffer and scrolls the end of the line back leftwards.

1.188 Delete End of Line (Keyboard Function)

Deletes all inputs right to and under the cursor up to the end of the line.

1.189 Cut End of Line (Keyboard Function)

Removes all inputs right to and under the cursor and copies the removed characters to the Yank buffer.

1.190 Delete Start of Line (Keyboard Function)

Deletes all inputs left to the cursor position and scrolls the remaining line backwards.

1.191 Cut Start of Line (Keyboard Function)

Removes the inputs left of the cursor and copies these characters to the Yank buffer. The remaining line is then scrolled backwards.

1.192 Delete End of Display (Keyboard Function)

Deletes all characters at, right to and under the cursor position.

1.193 Form Feed (Keyboard Function)

Clears the complete lower display buffer and places the cursor in the upper left corner of the window. If the window is in RAW mode , a "Form Feed" character ASCII 0C is send to the receiver.

1.194 Clear Screen (Keyboard Function)

Erases the complete lower display buffer and places the cursor in the upper left corner of the window. If the window is in Shell mode and a shell is currently waiting for inputs, the shell prompt is requested again.

1.195 Cut (Keyboard Function)

Removes the currently marked block from the display buffer and copies it to the clipboard.

1.196 Copy (Keyboard Function)

Copies the currently marked block to the clipboard.

1.197 Paste (Keyboard Function)

Inserts the clipboard contents into the input stream as if you've typed the clipboard contents yourself with the keyboard.

1.198 Hide (Keyboard Function)

Removes the block mark from all characters in the block , i.e. un-highlights the block.

1.199 Select All (Keyboard Function)

Selects the complete contents of the display buffers and marks them as a block .

1.200 Copy Quiet (Keyboard Function)

Copies the currently marked block to the clipboard, but unlike the Copy function, this does not hide the marked block

1.201 Reset (Keyboard Function)

Erases the lower display buffer and resets the mode flags of the terminal emulation to their preferences values and restores the colors selected in the preferences setting of the window.

1.202 Full Reset (Keyboard Function)

Erases all display buffers completely, clears the command history and restores the mode flags and the colors of the window to the selected preference values.

1.203 Toggle ESC (Keyboard Function)

Toggles the ESC flag on/off. If a control function is found in the input stream with the ESC flag enabled, the CSI sequence of this control function is inserted literally into the line instead of getting interpreted by the keyboard parser. This is especially useful to insert control characters into the input stream, as for example as control characters for a string to be printed by "echo".

This function works identically to the ESC key of the Atari 8 bit computers.

1.204 Toggle NumLock (Keyboard Function)

Toggles the keyboard qualifier "NumLock" on/off. Some keyboard functions are only recognized if "NumLock" is enabled. This is especially useful for the keys of the numeric keypad: These keys could be bound to work as cursor keys if "NumLock" is enabled.

1.205 Toggle Overwrite (Keyboard Function)

Toggles the overwrite flag on/off. If overwrite is enabled, characters typed on the keyboard will write over the characters already in the display buffer instead of getting inserted.

1.206 Suspend (Keyboard Function)

Suspends, i.e. stop all output. This is also known as "XOFF" function.

1.207 Resume (Keyboard Function)

Resumes stopped output. This is also known as "XON" function.

1.208 Abort Expansion (Keyboard Function)

Aborts a currently running TAB expansion . This is implicit for most keyboard functions anyhow.

1.209 Scroll to Cursor (Keyboard Function)

In case the cursor has been scrolled out of the window, this function scrolls the window back such that the cursor gets visible again. This is implicit for most keyboard functions anyhow.

1.210 Rewind History (Keyboard Function)

Removes all user inputs from the current line and rewinds the command history .

1.211 Yank (Keyboard Function)

Inserts the contents of the Yank buffer at the cursor position.

1.212 Generate EOF (Keyboard Function)

Dependent on the Call macro to close window flag, this selects either one of the two "Close" system macros or sends an "End Of File" condition to the receiver. This function is invoked by the close gadget of the window, too.

1.213 Display Beep (Keyboard Function)

Beeps the screen. Dependent on the settings of the "Sound" system preferences, this may also cause an audible beep.

1.214 Toggle Pause (Keyboard Function)

This toggles the output on and off; if the function is invoked the first time, output will be stopped. On the second time, output will resume.

1.215 Help (Keyboard Function)

Invokes the "Help" system macro , or sends the Help CSI sequence if the window is in RAW Mode .

1.216 Fork New Shell (Keyboard Function)

Forks a new shell in the current window, sending the currently running job to background. This is the keyboard function responsible for the Ctrl-Z feature of ViNCEd.

1.217 Insert CSI (Keyboard Function)

Inserts a literal CSI (ANSI 9B) character into the display buffer. This might be useful as an argument of the "echo" command to print a control sequence.

1.218 Insert ESC (Keyboard Function)

Inserts a literal ESC (ASCII 1B) character into the display buffer. This might be useful as an argument to the "echo" command to allow printing of control sequences.

1.219 The graphical Interface of SetVNC

If you load SetVNC without any arguments, with the command "SetVNC Prefs", or by clicking on the "SetVNC" icon in the "Prefs" drawer, a graphical user interface will be build.

Because the screen is much to small to hold all gadgets necessary to control ViNCEd, the settings is split into several pages , grouped by function. Most pages are itself split into a front page and one or more back pages which can be selected by the arrow gadgets " « " and " » " at the top right edge of each page. Left to these arrow gadgets is the "Help" button which invokes context sensitive help about the page shown. If these gadgets do NOT work, please check that the path to this guide, which can be found in the first system page is setup properly. If not, correct it with the string gadget on this page, or locate the guide using a requester by pressing the gadget left to it.

Additional to the "Help" gadget, each page has a row of four or three gadgets at the bottom, used to accept or discard the made changes. The second gadget from the right, labeled "To Window", is not present if the argument "Prefs" was used to invoke SetVNC as a preferences editor.

Save

Save the changes to a place where they will be loaded each time the system restarts and use the settings for all new windows to be opened.

WARNING: Pressing this button DOES NOT change the settings of the windows already open. Their settings are under control of the programs running in them, not of SetVNC!

Use

Use the settings for all new ViNCEd windows, but do not save. The settings will be reset when the system gets rebooted.

WARNING: Typing "Use" WILL NOT change the settings of the windows already open! IT WILL ONLY CHANGE THE DEFAULTS FOR NEW WINDOWS.

Pressing either "Use" or "Save" will also change the settings in the currently active window, as if "To Window" has been selected as well.

To Window

Install the settings in the window which was used to invoke SetVNC. This gadget is only present if "SetVNC" was NOT invoked as a preferences editor, i.e. the PREFS argument was NOT given.

It DOES NOT touch the settings of all other windows nor the settings of windows to be created.

Cancel

Aborts the editor and discards the changes; discards also all shell arguments .

The pages itself are selected with the "card-index" like gadget on top of the window. The page index is found in a different section .

1.220 The control pages of SetVNC

Here's the index of the "Pages" of the SetVNC program. More details about how they work can be obtained by following the links below, some general remarks about them are in the GUI interface section .

The About Page

The first Macros Page The second Macros page The third Macros page

The first Keyboard Page The second Keyboard Page

The first Edit Page The second Edit Page The third Edit Page The fourth Edit Page

The first Shell Page The second Shell Page The third Shell Page The fourth Shell Page The fifth Shell Page The sixth Shell Page

The first Window Page The second Window Page The third Window Page

The first Timing page

The first System Page The second System Page The third System Page The fourth System Page

1.221 Shell and Workbench Operation of SetVNC

The SetVNC program can be invoked from both the workbench and the shell. In the Shell operation , arguments are given on the command line, whereas in Workbench operation the ToolTypes of the SetVNC icon are parsed. Invoking SetVNC from the workbench is usually restricted to setup the global preferences of ViNCEd, whereas you may use SetVNC from the shell to adjust the ViNCEd settings of the window you invoked SetVNC from "on the fly".

Besides preferences and settings control, the SetVNC program does even more:

It loads the ViNCEd main library and replaces the standard CON: handler on request, it offers commands for job control and for loading and saving the various buffers of ViNCEd, as the command history and the review buffer . It loads this guide, too, if help is requested by pressing the Help key in the shell.

It can also be used to check whether the window SetVNC was invoked from is a ViNCEd window or not and is therefore useful in various scripts to enable special ViNCEd improvements.

Further details are found by following these links:

Workbench ToolTypes

Shell Arguments

Job Control

Buffer input/output

1.222 Workbench ToolTypes of the SetVNC program

The SetVNC program can also invoked from the workbench, mostly used as a preference editor. It recognizes the following tooltypes, which can be set in the usual way, by the "Information" item in the workbench menu:

WINDOW=path

Specify the window SetVNC should appear in. This tooltype MUST BE PRESENT, and it MUST be a path to a ViNCEd window. Details about the path are found in the window path section .

ACTION=args

Specifies the commands that SetVNC should perform. The "args" string is a list of shell arguments . They are passed to SetVNC as if it was called from the shell. The complete list of shell arguments is in the shell arguments section .

Most useful is the action list "LOAD GLOBAL MODIFY PREFS", which invokes SetVNC as a preferences editor.

HELPPATH=path

Specifies the complete path to this guide. In fact, this is the place where SetVNC looks first to find the guide, aside from a file "VNCGuide.path" in the ENVARC: drawer. This tooltype, as well as the environment variable gets adjusted if you change the guide path in the first system page . However, you shouldn't need to modify the environment variable as well as the tooltype by hand; it's setup correctly by the installation script and should be modified exclusively by SetVNC.

1.223 Shell Arguments of the SetVNC program

The power of the SetVNC command is the shell interface. It is possible to change most of the parameters of ViNCEd by this little program "on line" by using the build-in graphical user interface , or by invoking it from the shell for changing parameters in a script file. Except for preference adjustment, SetVNC manages, too, certain ViNCEd specific features as reading or writing the display buffers , or job control related functions.

Just an important note at this place:

Do NOT redirect the output of SetVNC to NIL:. This won't work. The output stream is needed by SetVNC to identify the ViNCEd window it is working in. If required, use the QUIET argument below.

The complete list of recognized arguments can be asked for with the command line

1.SYS:> SetVNC ?Return

as usual. The type of the argument is indicated by a forwards slash "/" following the argument:

/S is a switch argument. If present, it enables a certain feature. No parameters are required.

/K is a key word argument. It takes one or several sub arguments and their parameters to control a specific feature. Most noticeable are keyword arguments that take either the switches "ON/S" or "OFF/S" as sub arguments. These enable or disable certain features of ViNCEd, identical to those controlled by the GUI .

/N identifies arguments requiring numeric parameters.

(no slash) identifies arguments taking strings as parameters.

Here's now the list of all shell arguments. Just click on an item to see what this specific argument does:

QUIET/S

MOUNT/K (OVERRIDE/S AS) HELP/S

BACKGROUND/S FOREGROUND/K (OTHER/S CLI/N)

FREEPOINTER/K (ALL/S) SETCONSOLE/S

LOAD/K (PREFS/S GLOBAL/S WINDOW/S DEFAULT/S LAST/S FROM) PREFS/S

PUT/K (SCREEN HISTORY) GET/K (SCREEN HISTORY)

RESET/S DOSCURSOR/K (ON/S OFF/S)

CRINSERT/K (ON/S OFF/S) OVERWRITE/K (ON/S OFF/S)

WRAP/K (ON/S OFF/S) SMARTCLOSE/K (ON/S OFF/S)

SAFERCLOSE/K (ON/S OFF/S) DISABLEPROPX/K (ON/S OFF/S)

DISABLEPROPY/K (ON/S OFF/S) CLOSEREQ/K (ON/S OFF/S)

CLOSEQUEUE/K (ON/S OFF/S) CUTUSER/K (ON/S OFF/S)

REBUILDDELAY/K (ON/S OFF/S) AUTOCOPY/K (ON/S OFF/S)

NUMPADMODE/K (ON/S OFF/S) SCROLLBORDERS/K (ON/S OFF/S)

TYPEAHEAD/K (ON/S OFF/S) BOLDEXTCOLORS/K (ON/S OFF/S)

FORBIDICONIFY/K (ON/S OFF/S) CHUNKYPIXEL/K (ON/S OFF/S)

SHELLMODE/K (ON/S OFF/S) NODEFAULTCLOSE/K (ON/S OFF/S)

NOPRINTSCROLL/K (ON/S OFF/S) SMALLDISPLAY/K (ON/S OFF/S)

DOSERASE/K (ON/S OFF/S) DOSINSERT/K (ON/S OFF/S)

SHORTSCREENINSERT/K (ON/S OFF/S) AUTOPASTE/K (ON/S OFF/S)

DISABLESCROLL/K (ON/S OFF/S) DISABLEWRAP/K (ON/S OFF/S)

XTERMCURSOR/K (ON/S OFF/S) RIGIDCURSOR/K (ON/S OFF/S)

KEEPDOUBLES/K (ON/S OFF/S) SCROLLTOBOTTOM/K (ON/S OFF/S)

RAWSCROLLERS/K (ON/S OFF/S) VTMODE/K (ON/S OFF/S)

ANSIMODE/K (ON/S OFF/S) ANSIREVERSE/K (ON/S OFF/S)

ROWLOCK/K (ON/S OFF/S) UNDERLINE/K (ON/S OFF/S)

BLINKING/K (ON/S OFF/S) DISABLEMMB/K (ON/S OFF/S)

NOBSSTART/K (ON/S OFF/S) DONTPLACEREQUESTER/K (ON/S OFF/S)

BUFFERSIZE/K/N LOWERSIZE/K/N

UPPERSIZE/K/N CACHESIZE/K/N

MONITORID/K/N MONITOR/A

DELAYINTERVAL/K/N REBUILDTIME/K/N

SCROLLTIME/K/N BLINKSPEED/K/N

PATHONLYQU/K/N NAMEONLYQU/K/N

REQ_LEFTEDGE/K/N REQ_TOPEDGE/K/N

REQ_WIDTH/K/N REQ_HEIGHT/K/N

MODIFY/S SAVE/K (NOICONS/S PREFS/S GLOBAL/S WINDOW/S TO)

IFVNC/S

1.224 Job Control related arguments of SetVNC

The SetVNC program is not only the preferences editor of ViNCEd, it's, too, responsible for the job control . The following shell arguments are dedicated for this function:

SetVNC Background

Sends the current foreground job to background.

SetVNC Foreground <cli>

Takes one additional argument, the "Cli number", and sends the job related to this Cli number to foreground. The "Cli number" can be obtained by the "status" command.

The "Foreground" option is restricted to jobs running in the same ViNCEd window as the window you invoked SetVNC from. In all other cases, SetVNC will print a warning.

SetVNC Foreground other <cli>

Takes one additional argument, a "Cli number". As the argument before, this sends the job related to the given Cli number to foreground. However, this command is not restricted to jobs running in the same output window than the SetVNC program, it may be used to push jobs running in other windows to foreground, by using a "brute force" method. However, the use of the "other" keyword is discouraged and it should be used only in emergency situations since this operation is not 100% "water proof". It may crash the computer in (admittedly rare) conditions.

To avoid unnecessary typing, the foreground and background commands have been abbreviated in three scripts , named "fg", "bg" and "fork" which are more convenient for all-day purpose. For details about them, check the script section .

More details about the job control functions and its implementation, the console owner system , are found by following these links.

All other shell arguments of SetVNC are in a different section .

1.225 SetVNC Buffer I/O functions

SetVNC is not only the preferences editor of ViNCEd, it can be used, too, to save the ViNCEd internal buffers , as the command history and the review buffer . The following two shell arguments are reserved for saving and loading the buffer contents as an ASCII text file:

SetVNC Put History <file>

Takes one additional argument, a file name. This saves the contents of the history to a plain text file. Remember that you may, too, print the contents as text to the current window by using "*" as a file name.

(Side remark: As for ViNCEd, * is different to CONSOLE: Details are in a different section)

SetVNC Get History <file>

Takes again a file name as additional argument and loads a plain text file as history contents. The old contents of the history is lost.

SetVNC Put Screen <file>

This command saves the contents of the display buffer to the given file name. The output file will be a text file with embedded CSI sequences that describe the text rendering options. More details about the CSI sequences used are in the CSI and ESC sequences overview.

SetVNC Get Screen <file>

Loads the review buffer from the given file. Embedded CSI sequences will be interpreted accordingly, but not the full set of ViNCEd sequences is available in the load file, mainly for speed reasons. The main purpose of this command is not to display a file on the screen - use the standard "Type" command for that purpose - but to re-load a previously saved buffer contents, as created by "SetVNC Put Screen <file>". It will, unlike "Type", also restore all ViNCEd internal text attributes ; for example, whether the text was typed as user input or printed as output.

This command erases the old buffer contents completely and replaces it with the data from the given file.

All other shell arguments of SetVNC are in a different section .

1.226 The Format of the Preferences File

The ViNCEd settings are kept in the file ViNCEd.prefs in the ENVARC: and ENV: drawer and are read from that location by the vnc.library. Whereas it is recommended that this file is only changed by the SetVNC program, you may edit it with an editor of your choice as well - the file is a plain ASCII file.

The parser of the preferences file ignores all empty lines or lines starting with a semicolon, which are supposed to be comment lines. ViNCEd inserts some comments itself, just for your convenience - however, they are simply ignored when the file is read later on. A semicolon may not be put anywhere else except at the start of the line, and it will be recognized as a comment only in this position. Leading and trailing spaces of all lines will be removed as well - this might be of importance for settings that require a string as an argument, as the macro settings. In cases where this is required, the macros should be included in double quotes.

A setting itself consists of the NAME of the option to be set, an equal sign and the value of this option. The order of the options does not matter, but choosing a different order may slow down the preferences parsing process; the parser is optimized to read the settings in the same order as it writes it. Invalid or unknown options will be silently ignored, for forwards compatibility to future versions.

Each option requires an argument of a certain kind - the following argument types are used:

Boolean

A boolean on/off switch. Recommended are "on" and "off" to set this option, even though ViNCEd recognizes some more phrases.

Integer

An integral number, usually within a restricted interval. Values outside of this interval are ignored. The number should be given in decimal notation, even though ViNCEd recognizes hex (0x or $) numbers, binary numbers (%) or octal numbers (§) as well on input.

Hex value

Again an integer, usually without any restriction. The number should be given as hexadecimal number with leading 0x on input, even though ViNCEd recognizes other number formats as well.

Keyboard qualifier

A keyboard qualifier value. Legal qualifiers are

Ctrl : The Ctrl key Shift : Either Shift key Alt : Either Alt key Amiga : The Right Amiga key

No other qualifiers are allowed here. Especially, left and right Shift and left and right Alt keys are considered as equal.

Color specifier

Used to define ViNCEd internal colors. A color specifier consists of five words, separated by commas.

The first word is either "LOAD" or "NOLOAD" and defines whether the color value defined below is supposed to be loaded into one of the hardware registers if ViNCEd opens a private screen.

The next word is either "ANSI" or "NOANSI". If "ANSI" is given, this entry defines a standard ANSI color instead of a hardware register. If both, "ANSI" and "LOAD" are given, the color allocates an ANSI color with a higher priority.

The next three slots represent the red, green and blue components of the color, given as sixteen bit (word sized) fractional values in hexadecimal notation. The numbers must be given with a leading "0x" to identify them as hex values, even though ViNCEd may accept other number bases here as well on input. The number 0x0000 is minimal intensity, 0xffff is maximal intensity, 0x7fff is half intensity. The numbers are rounded for the available graphics hardware if required.

String

An ASCII string. Leading and trailing spaces are truncated by the preferences parser, but except that no interpretation is done on the string itself. Especially, a semicolon in these strings is treaded as a literal semicolon and not as a comment introducer. Other parts of ViNCEd allow certain strings to be enclosed in double quotes to include spaces, as for example the macro interpreter . However, this interpretation is not done by the parser itself.

Key

Defines a keyboard key sequence for definition of the keyboard functions . It consists of one or more qualifiers and the key itself, to be separated by blank spaces.

The following qualifiers are available:

Ctrl : the Ctrl key Shift : either Shift key Alt : either Alt key LShift : only the Left Shift key RShift : only the Right Shift key LAlt : only the Left Alt key RAlt : only the Right Alt key LAmiga : the Left Amiga key. This is the "Commodore" key on some keyboards. RAmiga : the Right Amiga key

As a special rule, "LShift RShift" means "Left OR Right", i.e. is equivalent to "Shift" alone, same goes for the "Alt" keys. For "Amiga", things are simpler. Presenting both means really you've to press both to get the function.

Another special qualifier is

Num : the function is only available with NumLock active.

This does, to stress that again, NOT MEAN that the keyboard key is a key on the numeric keypad. It means ONLY that the keyboard function is only available if "NumLock" is active.

"NumLock" is a keyboard qualifier similar to the "Caps Lock" qualifier, with the difference that there is no light on the keyboard to show its state. It is, however, treated exactly in the same way. The keyboard function that toggles this qualifier is, obviously, called Toggle NumLock and is usually bound to Alt Num[.

The final part of a keyboard key definition is the key itself. This key might be given in three different ways:

- as a hexadecimal number with leading "0x" defining the raw key code of the key to be used. The keyboard raw key codes can be found in the literature .

- as a string defining one of the special keyboard keys. The following strings are recognized:

Esc : the Escape key F1 to F10 : the function keys Backspace : the Backspace key Del : the Delete key Help : the Help key Tab : the TAB key Return : the Return key on the main keypad Space : the space bar Enter : the Enter key on the numeric keypad Up : the Cursor Up key Down : the Cursor Down key Left : the Cursor Left key Right : the Cursor Right key

The next "keywords" identify keys on the numeric pad. Do not confuse this with the "Num" qualifier which has a different meaning. Please note, too, that there is no space between the "Num" and the key name.

Num[Num] Num/ Num* Num7 Num8 Num9 Num- Num4 Num5 Num6 Num+ Num1 Num2 Num3 Num0 Num.

- and, as the third method, the single character which is printed on all "ordinary" keyboard keys. Such as "a" will identify the "a"-key. If a key is defined this way, the "Shift" qualifiers are ignored. Instead of using an explicit qualifier definition for "Shift", the "Shift" qualifier is implicitly defined by the ASCII character in the definition. For example, "a" will match the un-qualified "a"-key, whereas "A" is the "a" key together with Shift. For the same reason is "1" the un-qualified "1" key, whereas "!" is the "1" key together with Shift. This special rule holds only for the Shift qualifier, not for Alt or any other modifiers. Thus, to identify the key "a" with Shift and Alt, you need to type

Alt A

and not

æ or Shift Alt a

There is a tricky difference between the first two and the last method of defining a keyboard function. Whereas the first two methods identify a key by its raw key code, i.e. relate directly to the position of the key on the keyboard matrix, the last definition identifies a key by its ASCII value. This ASCII value might, however, depend on the national keyboard. To give an example: The key with the raw key code 0x15 is the "y" key on american keyboard, but the "z" key on the german keyboard. Thus, when you define a key by its hex raw key value, this will identify the same physical key on all national keyboards, but this key may have different functions. Whereas if you define a key in the third way, by its ASCII value, this definition might refer to a different physical key on different keyboards, but it is guaranteed that this key is always labeled with the same letter, even though its located at a different position. It is therefore recommend to use the third "ASCII match" method for defining keys except for functions using the special keys which do not depend on national keyboard definitions.

Now for the list of all available options in the preferences file; by following the links and pressing the "Contents" gadget of the browser above, you're brought to the SetVNC page that controls this specific feature:

Name of the option Type

Various flags:

REBUILDDELAY boolean DOSCURSOR boolean CRINSERT boolean OVERWRITE boolean WRAP boolean SMARTCLOSE boolean CLOSEQUEUE boolean CUTUSER boolean CHUNKYPIXEL boolean SHELLMODE boolean AUTOINDENT boolean NODEFAULTCLOSE boolean AUTOCOPY boolean SAFERCLOSE boolean FORBIDICONIFY boolean DISABLEMMB boolean CLOSEREQ boolean SCROLLTOBOTTOM boolean NOPRINTSCROLL boolean SMALLDISPLAY boolean DOSERASE boolean AUTOPASTE boolean DISABLESCROLL boolean DISABLEWRAP boolean DOSINSERT boolean VTMODE boolean ROWLOCK boolean UNDERLINE boolean BLINKING boolean XTERMCURSOR boolean NOBSSTART boolean ANSI-MODE boolean ANSIREVERSE boolean NUMPADMODE boolean BOLDEXTCOLORS boolean SHORTSCREENINSERT boolean SCROLLBORDERS boolean TYPEAHEAD boolean DISABLEPROPX boolean DISABLEPROPY boolean RAWSCROLLER boolean RIGIDCURSOR boolean KEEPDOUBLES boolean DONTPLACEREQUESTER boolean

Buffer sizes:

BUFFERSIZE integer between 5 and 4096 UPPERSIZE integer between 64 and 4096 LOWERSIZE integer between 128 and 4096 CACHESIZE integer between 1 and 256

Timing values:

DELAYINTERVAL integer between 50 and 1000 REBUILDTIME integer between 25 and 1000 SCROLLTIME integer between 5 and 500 BLINKSPEED integer between 100 and 1000

Miscellaneous:

MONITORID hex value

PATHONLYQU keyboard qualifier NAMEONLYQU keyboard qualifier

Colors:

CURSORCOLOR color specifier

up to sixteen times: COLOR color specifier

TAB expansion settings:

TAB_FILE_PRI integer between -128 and 127 TAB_EXEC_PRI integer between -128 and 127 TAB_SCRIPT_PRI integer between -128 and 127 TAB_PATH_PRI integer between -128 and 127 TAB_COMMAND_PRI integer between -128 and 127 TAB_RESIDENT_PRI integer between -128 and 127 TAB_INFO_PRI integer between -128 and 127 TAB_DEVICE_PRI integer between -128 and 127 TAB_ASSIGN_PRI integer between -128 and 127 TAB_VOLUME_PRI integer between -128 and 127 TAB_DIRECTORY_PRI integer between -128 and 127 TAB_DOUBLEREQ integer between -128 and 127 TAB_FULLEXPAND integer between -128 and 127 TAB_VNCREQUESTER integer between -128 and 127 TAB_AMBIGREQ integer between -128 and 127

TAB expansion settings for the short path:

SRT_FILE_PRI integer between -128 and 127 SRT_EXEC_PRI integer between -128 and 127 SRT_SCRIPT_PRI integer between -128 and 127 SRT_PATH_PRI integer between -128 and 127 SRT_COMMAND_PRI integer between -128 and 127 SRT_RESIDENT_PRI integer between -128 and 127 SRT_INFO_PRI integer between -128 and 127 SRT_DEVICE_PRI integer between -128 and 127 SRT_ASSIGN_PRI integer between -128 and 127 SRT_VOLUME_PRI integer between -128 and 127 SRT_DIRECTORY_PRI integer between -128 and 127 SRT_DOUBLEREQ integer between -128 and 127 SRT_FULLEXPAND integer between -128 and 127 SRT_VNCREQUESTER integer between -128 and 127 SRT_AMBIGREQ integer between -128 and 127

TAB devices expansion settings:

DEV_FILE_PRI integer between -128 and 127 DEV_EXEC_PRI integer between -128 and 127 DEV_SCRIPT_PRI integer between -128 and 127 DEV_PATH_PRI integer between -128 and 127 DEV_COMMAND_PRI integer between -128 and 127 DEV_RESIDENT_PRI integer between -128 and 127 DEV_INFO_PRI integer between -128 and 127 DEV_DEVICE_PRI integer between -128 and 127 DEV_ASSIGN_PRI integer between -128 and 127 DEV_VOLUME_PRI integer between -128 and 127 DEV_DIRECTORY_PRI integer between -128 and 127 DEV_DOUBLEREQ integer between -128 and 127 DEV_FULLEXPAND integer between -128 and 127 DEV_VNCREQUESTER integer between -128 and 127 DEV_AMBIGREQ integer between -128 and 127

TAB directory expansion settings:

DIR_FILE_PRI integer between -128 and 127 DIR_EXEC_PRI integer between -128 and 127 DIR_SCRIPT_PRI integer between -128 and 127 DIR_PATH_PRI integer between -128 and 127 DIR_COMMAND_PRI integer between -128 and 127 DIR_RESIDENT_PRI integer between -128 and 127 DIR_INFO_PRI integer between -128 and 127 DIR_DEVICE_PRI integer between -128 and 127 DIR_ASSIGN_PRI integer between -128 and 127 DIR_VOLUME_PRI integer between -128 and 127 DIR_DIRECTORY_PRI integer between -128 and 127 DIR_DOUBLEREQ integer between -128 and 127 DIR_FULLEXPAND integer between -128 and 127 DIR_VNCREQUESTER integer between -128 and 127 DIR_AMBIGREQ integer between -128 and 127

TAB icon (.info) expansion settings:

INF_FILE_PRI integer between -128 and 127 INF_EXEC_PRI integer between -128 and 127 INF_SCRIPT_PRI integer between -128 and 127 INF_PATH_PRI integer between -128 and 127 INF_COMMAND_PRI integer between -128 and 127 INF_RESIDENT_PRI integer between -128 and 127 INF_INFO_PRI integer between -128 and 127 INF_DEVICE_PRI integer between -128 and 127 INF_ASSIGN_PRI integer between -128 and 127 INF_VOLUME_PRI integer between -128 and 127 INF_DIRECTORY_PRI integer between -128 and 127 INF_DOUBLEREQ integer between -128 and 127 INF_FULLEXPAND integer between -128 and 127 INF_VNCREQUESTER integer between -128 and 127 INF_AMBIGREQ integer between -128 and 127

TAB alternate expansion settings:

ALT_FILE_PRI integer between -128 and 127 ALT_EXEC_PRI integer between -128 and 127 ALT_SCRIPT_PRI integer between -128 and 127 ALT_PATH_PRI integer between -128 and 127 ALT_COMMAND_PRI integer between -128 and 127 ALT_RESIDENT_PRI integer between -128 and 127 ALT_INFO_PRI integer between -128 and 127 ALT_DEVICE_PRI integer between -128 and 127 ALT_ASSIGN_PRI integer between -128 and 127 ALT_VOLUME_PRI integer between -128 and 127 ALT_DIRECTORY_PRI integer between -128 and 127 ALT_DOUBLEREQ integer between -128 and 127 ALT_FULLEXPAND integer between -128 and 127 ALT_VNCREQUESTER integer between -128 and 127 ALT_AMBIGREQ integer between -128 and 127

Requester dimensions:

REQ_LEFTEDGE integer between -2048 and 2048 REQ_TOPEDGE integer between -2048 and 2048 REQ_WIDTH integer between 0 and 2048 REQ_HEIGHT integer between 0 and 2048

Macros: up to ten times:

MACRO string

System macros: up to five times:

SYSTEMMACRO string

Other system strings:

RUN_NEW_SHELL string ICON_PATH string ICON_TITLE string QUIT_PROGRAM string DEFAULT_FONT string DEFAULT_PATH string

Buttons: up to ten times:

BUTTONMACRO string BUTTONTITLE string

Keyboard:

CURSOR_LEFT key CURSOR_RIGHT key CURSOR_UP key CURSOR_DOWN key HISTORY_UP key HISTORY_DOWN key SEARCH_PARTIAL_UPWARDS key SEARCH_PARTIAL_DOWNWARDS key SEARCH_HISTORY_UPWARDS key SEARCH_HISTORY_DOWNWARDS key HALF_SCREEN_LEFT key HALF_SCREEN_RIGHT key HALF_SCREEN_UP key HALF_SCREEN_DOWN key SCROLL_UP key SCROLL_DOWN key SCROLL_HALF_SCREEN_UP key SCROLL_HALF_SC
key TO_LEFT_BORDER key TO_RIGHT_BORDER key TO_TOP_OF_SCREEN key TO_BOTTOM_OF_SCREEN key PREV_WOR
key NEXT_WORD key PREV_COMPONENT key NEXT_COMPONENT key HOME key END key

SEND_INPUTS key SPLIT_LINE key INSERT_^J key SEND_COMPLETE_LINE key LINE_FEED key

TAB_FORWARDS key TAB_BACKWARDS key

EXPAND_PATH key EXPAND_BACKWARDS key EXPAND_SHORT key EXPAND_SHORT_BKWDS key EXPAND_DEVICES
key EXPAND_DEVS_BKWDS key EXPAND_DIRS key EXPAND_DIRS_BKWDS key EXPAND_ICONS key EXPAND_ICONS_BK
key EXPAND_ALT key EXPAND_ALT_BKWDS key

SEND_^C key SEND_^D key SEND_^E key SEND_^F key SEND_^C_TO_ALL key SEND_^D_TO_ALL key SEND_^E_TO_ALL key SEND_^F_TO_ALL key

DELETE_FORWARDS key DELETE_BACKWARDS key DELETE_FULL_LINE key CUT_FULL_LINE key DELETE_INPUTS
key CUT_INPUTS key DELETE_WORD_FWDS key CUT_WORD_FWDS key DELETE_WORD_BKWDS key CUT_WORD_BKW
key DELETE_COMPONENT_FWDS key CUT_COMPONENT_FWDS key DELETE_COMPONENT_BKWDS key CUT_COMPON
key DELETE_END_OF_LINE key CUT_END_OF_LINE key DELETE_START_OF_LINE key CUT_START_OF_LINE key DELETE_END_OF_DISPLAY key FORM_FEED key CLEAR_SCREEN key

CUT key COPY key PASTE key HIDE key SELECT_ALL key COPY_QUIET key RESET key FULL_RESET key

TOGGLE_ESC key TOGGLE_NUMLOCK key TOGGLE_OVERWRITE key SUSPEND key RESUME key ABORT_EXPANSION key SCROLL_TO_CURSOR key REWIND_HISTORY key YANK key GENERATE_EOF key DISPLAY_BEEP key TOGGLE_PAUSE key HELP key FORK_NEW_SHELL key INSERT_CSI key INSERT_ESC key

1.227 The About Pages

The about pages....

...presents my logo in the middle. (No, that’s not me, I don’t have the right ears...)

In the bottom, you find

the "About" gadget, that invokes this guide if setup correctly. If not, locate it in the System front page .

To the right of this gadget, two "arrow gadgets « and »". Both will move you to the second "About" page. It looks almost like the first, except for the middle line of gadgets.

The next line of gadgets are used to re-load previously used preferences into the editor. These are usually four gadgets unless SetVNC was called with the PREFS argument which tells SetVNC not to modify the local preferences of the window it was invoked from, but the global preferences used for all windows to be opened.

More functions to load and save the preferences are on the second "About" page.

The meaning of the gadgets on the first page, from left to right is:

Last Saved

Load the last saved preferences, found on your HD in ENVARC:ViNCEd.prefs.

Last Used

Restore the previously used global settings, undo all changes made so far. These settings are automatically loaded by SetVNC if the PREFS argument was found on the command line .

Window

Re-load the settings from the window which SetVNC invoked from and undo all changes. This gadget is only available if no PREFS argument was given on the shell command line , and loading the preferences from the window is the default in this case.

Default

Load the "factory defaults" of ViNCEd.

If you moved to the second "About" page, this line contains only two gadgets, namely:

Save to File...

Presents a file requester to select a file to save the currently made preferences to.

Load from File...

This loads the preferences from a given file, again asking with a requester.

The gadgets in the last line are present in each page and described in a different section . They are used to save or install the changed settings.

1.228 The Macros Page 1 and 2

The purpose of the "Macros Pages" is to define the menu and title bar macros and buttons ViNCEd offers.

The first two macros pages define the macros that appear in the Macros Menu using the keyboard shortcuts Amiga 0 to Amiga 9.

The first page defines the macros 0 to 4, the second number 5 to 9, the arrow gadgets "«" and "»" in the right bottom corner of the window toggle between the pages.

If you want to change the contents of a macros, just type it in one of the gadgets; the text in front of them tells you the number of the macro, and thus the key used to invoke it.

To find out more about macros, study the macros section .

1.229 The Macros Page 3

Similar to the first two Macros Pages , this page is used to define the buttons that appear in the window title of a ViNCEd window. Except for their different appearance, they work like macros, but require a "button title". It's this title which appears later on in the window title, not the macro body itself.

The top two string gadgets are used to adjust the title and the body of it. Just type them here.

Since the number of buttons is not fixed, only one button at a time is shown. Use the "Prev" and "Next" gadgets to edit the previous or the next button. The last available button will be always cleared for you, left empty to allow the input of an additional button - provided that the limit of maximal eight buttons is not yet reached.

If you want to remove a button, click the "Remove" gadget on the left side, the "Insert" button beneath let's you insert a new clear button BEFORE the button currently seen. Again, provided that the limit of eight buttons is not exceeded.

However, it is NOT legal to leave a button with an empty title. If this happens, SetVNC beeps the screen and you have to enter a title, or remove the button completely.

If you want to find out more closely how buttons and the related macros work, check the macros section .

1.230 The Keyboard Page 1

The two keyboard pages are used to setup the keyboard configuration of a ViNCEd window. Hence, they are used to setup the binding of keys to functions .

Using the first page, you select the key to define, the second page selects then the function.

by ASCII value

If this gadget is checked, the key is identified by the ASCII character it generates, not by the raw keymap code. It is rather essential to understand the difference since besides the ViNCEd keyboard definition, there's still the keyboard definition of the Os that maps keys to characters.

To give an example: The keys "y" and "z" are interchanged on the german keyboard. If the english keymap is active and you bind a function to the "z" key, it depends on this gadget what happens if the keyboard is later on changed to german. If this flag is set, the function "moves along" with the keyboard keys and is now available thru a different key - namely the former "y" key - which, however, still produces the same character, "z". If this gadget is not set, the function is bound to the real, physical key, and the same physical key will generate the same function, regardless of the keyboard definition of the user - even when the meaning of the keys should change. Thus, the former is some kind of "bind by meaning", whereas the later is "bind by hardware".

It is usually a wise idea to bind "keys by meaning", i.e. to leave this gadget checked. This does, however, work only if the key to be bound generates a single, alphanumerical character - these are usually all "white" keys on the keyboard. Function keys and control keys like Return or Backspace can't be bound this way. However, this doesn't matter since their meaning is fixed and does not depend on a national keyboard.

Key

Type here the key you want to define. Remember: Only alphanumerical keys will be accepted in case the "by ASCII value" gadget on the left hand side is checked. This gadget changes, too, slightly the function of the key definition:

If it is not checked, SetVNC will ignore all qualifiers at the time you type the key. They have to be selected with the gadgets below.

Almost the same rule holds if the "by ASCII value" gadget is checked, except for the "Shift" qualifier which matters and MUST be pressed at the time the key is selected by typing it.

Left Shift

If checked, the Left Shift key must be pressed together with the key selected above to invoke the function.

Right Shift

If checked, the Right Shift key must be pressed together with the key selected above to invoke the function.

If both, left and right Shift are checked, it won't matter whether the left or the right shift key is pressed along with the key, i.e. the both gadgets are combined by a logical "or" and there won't be a difference between "left" and "right shift" at all. If only one of these two gadgets are checked, then IT DOES matter. In this case, the keyboard function is available if and only if the corresponding shift key is pressed.

These two gadgets are disabled if the "by ASCII value" gadget above is checked. The "shift" qualifier is then determined by a different mechanism. It must be pressed along with the key to be defined at the time you type in this key in the "Key" gadget above.

Left Alt

Selects the Left Alt key as qualifier.

Right Alt

Ditto, but the Right Alternate key.

If both, left and right Alt are selected, this means that the keyboard function to be defined doesn't care about whether the left or the right alt key was used to invoke it; thus, these two gadgets are coupled by an "or". If only one of them is checked, the keyboard function will react only if the corresponding alt key is used - left and right WILL matter.

Left Amiga

This selects the Left Amiga Key as keyboard qualifier. This is the "Commodore" key on some keyboards.

Right Amiga

This selects the Right Amiga Key as qualifier. This qualifier is usually used for menu operations, check if the function you're going to define doesn't conflict with one of the menu keys. The menu will usually take precedence unless no menu is attached to the ViNCEd window.

If both, left and right Amiga keys are checked, you've really to press both keys to invoke the function (which is kind of a hard trick, or at least unconvenient). The gadgets are combined by "and", not "or".

Ctrl

This selects the Ctrl key as qualifier.

NumL only

The function is only available if the "NumLock" qualifier is active.

This does, to stress that again, NOT MEAN that the keyboard key is a key on the numeric keypad. It means ONLY that the keyboard function is only available if "NumLock" is active, even though this makes mostly sense for, but is not limited to keys on the numeric pad.

"NumLock" is a keyboard qualifier similar to the "Caps Lock" qualifier, with the difference that there is no light on the keyboard to show its state. It is, however, treated exactly in the same way. The keyboard function that toggles this qualifier is, obviously, called Toggle NumLock and is usually bound to Alt Num[.

As soon as you've completely defined the key to bind the keyboard function to, go to the second keyboard page to select this function.

1.231 The Keyboard Page 2

After selecting a proper key to be defined on the first keyboard page , this page is used to select a function for the key, or to check the current keyboard definition.

The left hand side of the page is just a big "list view" gadget, to select the functions from. The topmost selection is "(undefined)", meaning that the key should not be bound to any function and should operate in its "usual" way. Not all function keys need to be defined explicitly, some do already have meanings by the Os - for example, you need not to bind the Cursor Keys or the Return key, provided you like the default operations of the keys. Same goes for the Backspace and the Delete keys, as well as some other "basic" functions which are always available, even at unbound keys; for example, the "Break" keys Ctrl C to Ctrl F are of this kind.

Thus, you'll be able to work with ViNCEd even with a broken or corrupt preferences file.

There's a complete list of all keyboard functions available, together with the explanations what these functions do and what they could be used for.

The right hand side of the page contains a box showing the current key binding. Whenever you select a function from the left hand list, the keys that invoke these functions are shown in the right hand box. In case more than one key is bound to the selected function, the gadgets "« Prev Key" and "Next Key »" select one binding after another. If the function isn't bound to any key, this field is just left blank. Please note that you've to use the Keyboard Page 1 to change the binding, you won't be able to enter keyboard sequences into this box; moreover, just browsing with these gadgets doesn't perform any change on the definitions made so far.

Accept

Don't forget to press this button to perform the definition, or everything is lost. It will accept your definition and bring you back to the first keyboard page .

1.232 The Edit Page 1

This is the first edit page, used to define various cursor control related settings. Please follow the links to find out more about these settings.

Unrestricted cursor movement Don't scroll into the border

XTerm/CON: cursor mode Rigid XTerm cursor

Numeric keypad cursor control

1.233 The Edit Page 2

This is the second edit page. It is used to setup the cursor shape and properties, and some clipboard and block related settings. Please follow the links to find out more about these settings.

Underline cursor Blinking cursor

Don't write printed text into clipboard Implicit copy after text marking

Disable middle mouse button

1.234 The Edit Page 3

The third page controls the ANSI coloring features, and the delete, backspace and overwrite controls. Please follow the links below to find out more about the various settings

ANSI colors as default Inverse ANSI coloring

Standard CR insertion at start of line Disable BS at start of line

Overwrite mode

1.235 The Edit Page 4

The fourth edit page controls miscellaneous settings, the type ahead buffer and the window alignment on resize. Please follow the links to find out more about the flags here.

Invisible type ahead Keep bottom of window adjusted

1.236 The Shell Page 1

The first shell page controls how and when ViNCEd closes its windows, and selects the shell mode. Please follow the links to find out more about the settings on this page.

Call macro to close window Prevent accidental window closing

Enable close requester Use shell mode by default

Don't send EOF until everybody waits

1.237 The Shell Page 2

The second shell mode controls the history buffer, the TAB expansion cache and the RAW mode scrollers .

History buffer wraps around

Keep duplicates in the history

Enable scrollers in RAW mode

Cached directories

1.238 The Shell Page 3

The third shell page controls various flags of each of the six pairs of TAB expansion keyboard functions. Which pair is currently edited is printed directly below the headline of this page and can be selected by the "« Prev" and "Next »" gadgets right to it.

Here’s the list of options on this page; please follow the links to find out more about what these do:

Double TAB Requester First TAB Expands fully

Requester if expansion is ambiguous Add ViNCEd matches to the requester

More about the TAB expansion in general is available in the form of a tutorial , there’s also a list of all TAB related settings available.

1.239 The Shell Page 4

The fourth shell page is used to setup the priority ViNCEd assigns to various objects found on a TAB expansion search. The entries of higher priority are shown earlier, on top of the TAB expansion list; hence, you need less keyboard presses to reach them. Objects with a priority of -128 or below aren’t added to the list at all and will be dropped. That means specifically that you may disable certain object classes completely by assigning them a priority of -128.

There are six sets of priorities, one for each pair of a TAB expansion keyboard function . Which pair is currently edited is shown right below the headline of this page, and it can be selected with the "« Prev" and "Next »" gadgets right to this headline.

Here’s now the list of all available priorities; please follow the links to find out more about which objects they control:

Files Dirs Icons Devices Assigns Volumes Path C: Dir Resident Scripts Executables

Your priorities get "adjusted" a bit if you search for a directory or a device explicitly, i.e. the template ends either with a forwards slash "/" or with a colon ":". In this case, all other file types except directories or devices/assigns/volumes are ignored and the priority of the file type looked for is eventually raised from -128 to -127. This means especially that even if you disabled directories explicitly, searching for an object ending with "/" will actually match something.

Please note that most files on an Amiga device will match the "Executables" category, NOT the "Files" category. This is because the "e" bit of most files, indicating an executable, is usually set.

To find out more about the TAB expansion, consider reading the tutorial or check the list of all available TAB expansion settings.

1.240 The Shell Page 5

The fifth shell page controls the qualifiers of the icon drop feature of ViNCEd.

If you hold one of the keys listed on this page while you’re dropping an icon of the workbench in the ViNCEd window, only the path or the name of the icon will be inserted.

The left hand gadgets control which keys must be pressed to insert only the name of the icon, the right hand side is used to specify the keys for inserting the path only.

Please note that ViNCEd does not distinguish between the left and right Shift or Alt key here, each of them will work.

More than one gadget can be checked if you want to enable these functions with pressing more than one key at once. Please avoid selecting the same qualifier on both sides, or selecting no qualifier at all. This is valid for ViNCEd, but may result in disabling the regular icon drop feature.

1.241 The Shell Page 6

This page is used to setup the position and the size of the TAB expansion requester.

Please follow the links for details:

Left Edge Top Edge

Width Height

Do not place file requester

This flag must be disabled, or the requester dimensions above will be ignored. This is a workaround against a feature of the "ReqChange/ReqTools" program that interprets the requester positions in a different way.

1.242 The Window Page 1

The first window page page is used to setup the size of the review and history buffers , and controls the default monitor and the default font used by ViNCEd.

History lines

Enter here the number of lines kept in the command history which holds all previous commands entered. Using the History Up and History Down

keyboard functions , these inputs can be again made visible. This must be a number between 5 and 4096.

Upper display size

Choose the number of lines hold in the upper display buffer . This buffers keeps lines that are scrolled off the top edge window. You can redisplay them by using the scroller on the right side of the window, or by scrolling into this buffer using the cursor movement keyboard functions .

As a rule of thumb, the upper display buffer should be half the size of the Lower display buffer, or bigger. The minimal possible size is 64, the maximal size is 4096 lines. If the upper display buffer overflows, which is usual when scrolling big directory lists, the topmost line gets lost. So if your listing does not fit, enlarge this buffer.

Lower display size

As you already might have thought, this is the number in the lower display buffer , which keeps the lines scrolled off the lower edge of the window and the text in the window itself. This buffer should be AT LEAST as big as the number of lines visible in the biggest possible window, values can range here from 128 to 4096. But I advice you to choose the size of the lower buffer at least twice the size of the upper buffer, since many lines go in this buffer if you scroll backwards to see the top of the window. If the buffer is too small, your latest outputs will be lost when scrolling back to the bottom of the text.

Monitor

This gadget is used to setup the default monitor ID when ViNCEd opens its own screen, using some special window path qualifiers.

The left hand button gives a screen mode requester if available - please select here your preferred monitor for ViNCEd screens.

The string gadget at the right side must be used if no screen requester is available (like for people still using the 1.3 or 2.0 workbench). Hex notation using "0x" or "$" is valid here, but the upper sixteen bits (the monitor ID itself) are ignored if you run ViNCEd on a 1.3 workbench.

Font

Select here the default font to be used in ViNCEd windows; this is NOT the font used for the menus which is taken from the screen the ViNCEd window appears on. It's the font used used for text in the window itself. Pressing the button yields a standard font requester, please choose a non-proportional font for ViNCEd here.

The string gadget next to the button is useful for setting up the font manually if no font requester is available. Enter here the name of the font without ".font", followed by a "." and the size as a decimal number. Hence, the string "topaz.8" will select the topaz font, size eight as the default.

This setting can be overridden by the FONT argument in the window path specified for a ViNCEd stream.

If you leave this gadget empty, the system default font will be used here.

Please note that the screen default font (and hence the font used for the menus) can not be selected with this gadget! This is up to the screen on which ViNCEd opened its window. The screen font of custom screens is currently not under control of ViNCEd and can be changed thru the system font preferences only.

If you want to find out how to setup the colors of the ViNCEd custom screens, read the next page , too.

1.243 The Window Page 2

This page is completely dedicated for setting up the colors of ViNCEd custom screens or for the ANSI pens. ViNCEd will open on a custom screen if some of the arguments in the window path say so; the colors defined on this page will then be loaded into the hardware. The ANSI colors define, however, which text rendering color is assigned to which screen pen, if this feature is enabled - which is either done by the third edit page or the ANSI window path argument; unlike the standard console, ViNCEd text rendering pens are not restricted to the first eight pens in a palette. ANSI colors can then be used even on non-custom screens: ViNCEd will try to allocate a public pen which matches the desired color most closely. More about ANSI colors is in a separate section .

ViNCEd holds sixteen color definitions, plus one selectable cursor color. To select the color you'd like to edit, use the "« Prev" and "Next »" buttons under the color sample. The cursor color is "left to" the color index #0, and is indicated by a "Cc" under the color sample box.

To edit one color, you have to select first the usage of this color with the two checkmark gadgets near the bottom of the window:

Load register

Try to load this color into the shown hardware register.

Define ANSI mapping

Don't define a hardware register, but the default color for the ANSI mode. The color index is in this case not the hardware register number, but the ANSI pen number. This flag works also for public and workbench screens, but doesn't load a color register unless no other matching pen was found.

Since the color setup can be a tricky business, there's a separate section available just explaining these two flags you should check.

The sliders

Checking either or both check marks makes the sliders at the left hand side of the page available. They are used to setup the red, green and blue component of the color. A color box at the right side of the page displays a sample of this color.

REMARK: It might happen that you can't see a sample of the color beneath the sliders cause all screen colors are allocated by other programs. In this case SetVNC tries to allocate a hardware sprite to display an additional color. This will yield to a rather strange behaviour of this color box. For example, it might "jump" out of the window when moving the window around, or might disappear if you activate a different window. This is due to hardware limitations! It might even fail to appear at all, either if no hardware sprite or no DMA time is available! This might happen with hacked monitor drivers or overscanned screens. Please note that it is hard enough to display an additional color on a four or eight color workbench! No problems should arise on Gfx boards cause they offer usually enough free pens that are used in this case instead.

1.244 The Window Page 3

The purpose of this page is to setup the defaults for which gadgets should be added to a VinCEd window. Please follow the links below to find out more about the flags here:

Don't add close gadget by default Don't add iconify gadget by default

Disable horizontal scroller Disable vertical scroller

1.245 The Timing Page 1

The timing of ViNCEd is setup by this page. This is not only the blink speed of the cursor - if you turned that on - but also two delay rates for refresh and scrolling.

All timings in this page are measured in milliseconds, that is a thousands of a second. The screen of a usual TV is refreshed every 20 or 17 milliseconds, depending on the system, PAL or NTSC. Values can be entered as the number (in "ms"), or chosen with the slider right to the gadget.

Please follow the links below to find out more about what these values mean:

Cursor blink speed Vertical scroll threshold

Rebuild delay Intuition delay

and finally, there's one checkmark-gadget that enables ViNCEd "turbo scrolling"; If this gadget is not checked, ViNCEd scrolls like the usual CON: handler and the timing values above are irrelevant.

Allow delayed window refresh

This check item can be found in the settings menu , too. It is called "Rebuild Delay" there.

1.246 The System Page 1

The first system page controls certain internal settings of ViNCEd you usually shouldn't mess around, i.e. hands off!. Editing this, and the other system pages is usually not required, except at installation time probably.

ViNCEd Guide Path

Enter here the complete path to the ViNCEd.guide that came with the ViNCEd archive (and you are currently reading). The gadget left to the path name will bring up a file requester, to help you locating the guide.

However, this requester will work ONLY, if the asl.library or the arp.library is available. While the first one comes with the Workbench, the last is public domain and is used as a replacement for the old workbench releases.

If you want to make the online guide system working, please make sure that:

o) The path of the guide is correctly setup in this gadget

o) The default tool of the guide itself is setup properly. To change it, locate the guides icon on the workbench, click it ONCE with the mouse and select "Info" from the workbench's icon menu. Enter as "Default Tool" in the upcoming window the COMPLETE PATH of your favorite guide browser; this should be "SYS:Utilities/Multiview" for Workbench 3.0 and up, "SYS:Utilities/AmigaGuide" for all other systems. "Multiview" comes with version 3.0 and 3.1 of the workbench, the "AmigaGuide" can be found in the AmiNet, for both versions of the OS, 2.0 and 1.3. However, the last one is a bit buggy (I had some crashes while testing it), and I recommend you to upgrade your system in that case.

Note: If you can't leave this page, make sure that the path to the guide is setup correctly. ViNCEd will not allow you to specify an invalid setting here.

The next four gadgets control various functions of ViNCEd; please follow the links to find out what they do:

Chunky pixel graphics Line break at right border

Inhibit horiz. scrolling by DOS output VT-220 compatibility mode

1.247 The System Page 2

The second system page controls various internal ViNCEd settings you usually do not want to change. Especially, selecting a flag here might easely cause compatibility problems, so just leave them alone.

To find out what these flags do, just follow the links below:

Destructive DEL and BS Insertion mode for DOS output

Notify DOS about paste Extended colors instead of bold

No insertion into border

1.248 The System Page 3

This page is used to setup various system macros needed by ViNCEd for internal control. They are somewhat similar to the user macros except that they are not available thru the macros menu in a ViNCEd window.

To find out more about how macros and the closely relied buttons are expanded, and about the special patterns and control sequences that can be used in them, read the macros section .

Quit shell

This macro is expanded by ViNCEd on a request to close the window if a shell is waiting for input in this window. This is only done if the Call macro to close window flag in the first shell page is selected, otherwise a "End Of File" condition is generated. Something like an "EndCLI" command should be placed here. The code $02 which is present in the default setting of this macro is the rewind history keyboard control sequence and erases the complete line before inserting the macro itself. A more sophisticated use of this macro would call a shell script which, for example, could erase temporary files created by a compiler session, and much more.

If you leave this macro empty, an EOF condition will be sent to the shell.

Quit program

This system macro will be expanded if the user presses the close gadget of a ViNCEd window while a program is being executed in it. It can be used to abort the command by sending a Ctrl-C or to send an EOF condition by leaving it empty.

New window

This macro gets expanded on selection of the "New Window" item in the project menu . It should bring up a new shell window, so something like a "NewCLI" command is the best choice. If you use a custom shell, enter here a similar command.

A second use of this macro would be to invoke a new editor window in a compiler session.

Fork new shell

This one is unique because it is not expanded into the keyboard buffer like all the other macros. Instead, it is sent directly to the shell segment for execution.

The macro, which is part of the job control of ViNCEd is used when a new shell is requested by a press of Ctrl-Z. It should bring up a new shell in the current window. If you use a custom shell, insert here a command similar to "NewShell WINDOW=*".

REMARKS: Since this macro is executed rather than inserted as key-presses, it must be terminated by a line feed sequence "\n" (ASCII 0A = 10) instead of a RETURN key press "\r" (ASCII 0D = 13).

The execution of this macro causes usually a re-execution of the S:Shell-Startup sequence. A usual problem with this is that you have to make this script re-entrant, i.e. stable for re-execution. One of the problems that may arise is the setting of pipe characters, namely the commands

Set _pchar | Set _mchar ||

Since on the second call, the definition of the pipe characters is interpreted as a pipe itself, this won't work. Change the lines to avoid the interpretation of the characters "|" and "||" by the shell:

Set _pchar "|" Set _mchar "||"

Note the double quotes!

Get help

Invoked if the user presses the "Help key" (or, to be precise, the key that generates the Help keyboard function) or selects the "Help" item of the project menu . This macro should bring up some help related to the current work. By default, the SetVNC program is invoked to display the ViNCEd help pages, i.e. this guide. See also the command line arguments of SetVNC for details which arguments to use.

A fine use of this macro would be to bring up the help pages of a compiler.

1.249 The System Page 4

The purpose of this page is to setup various system macros - look-alikes of the usual user macros which are required for internal operation of ViNCEd which do not appear in the macros menu . It's also used to define other string resources of ViNCEd. It's usually not required to change these settings except at installation time.

Edit settings

A system macro that gets expanded if the user selects the Settings item in the project menu. By default, this brings up the SetVNC preferences editor to change the settings of the window, but it might be quite useful to bring up a settings editor of a compiler, for example.

Icon Path

Enter a complete path to an icon (".info" - file) which should pop up if a ViNCEd window gets iconified. The position of the icon is also relevant, so it is probably a good idea to "UnSnapshot" it before using it for ViNCEd.

A ".info" MUST NOT be appended, that's done by ViNCEd automatically.

If this string is left black, ViNCEd will use its default icon, a simplified four color shell like window.

Icon title

This string specifies the default title used for the icon, i.e. the name that appears together with the icon on the workbench screen. If this string is left blank, the window title appears by default. However, a program might choose to setup its own title by the ESC sequence "ESC] 1; title BEL" that overrides this setting. The same control sequences as in the window title are available in this string as well, but since the icon title can't be changed too easely once ViNCEd has been iconified, it will be only expanded once, namely when the window gets iconified.

Default path

If ViNCEd was invoked just by the name of the handler and no window path at all, i.e. by using just "VNC:" as device specifier without additional arguments, this path will be used instead. It must consist of a window path like string except that the device specification in front of it must not be given, i.e. no leading "VNC:" or "CON:".

Note: Due to a "feature" of the 1.3 Mount command, this is disabled for older Os releases. Opening a window just with the handler name will (and has to!) fail for these releases.

1.250 QUIET (SetVNC argument)

Suppress verbose output, print only error messages. DO NOT REDIRECT SETVNC OUTPUT TO NIL: SINCE THE OUTPUT STREAM IS NEEDED TO FIND THE CONTROLLING VINCED WINDOW!

1.251 MOUNT/K (OVERRIDE/S AS) (SetVNC argument)

An argument that comes with additional subarguments. Its general use is to mount the ViNCEd handler explicitly, mostly to replace a different handler like CON: or RAW: by VNC. Everything else should be done with a standard mount icon in "Devs:Dosdrivers".

A line like

SetVNC mount override as CON

in your startup-sequence will replace for example the standard CON: handler by ViNCEd. Similarly,

SetVNC mount override as RAW

will replace the RAW: handler by the RAW version of ViNCEd, if you want to do so, too.

At least one sub argument is required for "MOUNT", namely the name of the handler that should be replaced by ViNCEd, as an argument to the "AS" keyword - which itself may be dropped. However, using only these arguments SetVNC will not try to replace an already loaded handler.

Since both the CON: and the RAW: handler ARE already loaded when the Os is booting, SetVNC must be called with the OVERRIDE parameter to unload the already loaded handlers and to replace them by ViNCEd. Therefore, an OVERRIDE almost always mandatory.

1.252 HELP/S (SetVNC argument)

Using this argument, SetVNC invokes the online help, meaning this guide.

To do so, it first reads the path of the ViNCEd guide from the ENV:VNCGuide.path variable, then checks the "Default Tool" of the guide file and runs this tool to display the guide.

All these details are usually setup as part of the installation procedure and you should not modify them by hand.

However, in case you move this guide to a different location, you should inform SetVNC about it and correct the internal information using the first system page .

In case you prefer a different browser for the guide, just use the workbench menu item "Info" on the guide icon and change the "Default Tool" of this guide. SetVNC will respect this setting for setting the guide.

1.253 BACKGROUND/S (SetVNC argument)

This is part of the job control arguments of SetVNC. It puts the current foreground job into background and makes the next available job the foreground job.

Details about the job control arguments are in a separate section .

1.254 FOREGROUND/K (OTHER/S CLI/N)

This is part of the job control arguments of SetVNC. It puts the job associated to the CLI number given for this argument to foreground , and releases the current job into background.

This works only for jobs running in the ViNCEd window the SetVNC command was invoked from, unless you specified the "OTHER" parameter of this argument as well. However, "OTHER" should be used in emergency situations only since it is not completely safe.

Details about the job control arguments are in a separate section .

1.255 FREEPOINTER/S (SetVNC argument)

This argument is a workaround against a design problem of the AmigaDos.

Calling

SetVNC FreePointer

will re-allow iconification and closing of AUTO style windows which might have been disabled by various programs, most notably the "More" browser. A workaround script has been put into the S: directory by the installation procedure using this command after running "More".

However, one "SetVNC FreePointer" will un-do the effect of exactly one "More" command (or to be precise, the effect of one ACTION_DISK_INFO). If you run "More" more than once, you might want to use the even stricter form

SetVNC FreePointer All

which will revert the side effects of ALL "More" commands at once.

There exists a documented method of obtaining information about the "intuition window" associated to a shell window. However, not each ViNCEd stream has to be attached to a window at all - for example, an AUTO style window which has not been opened yet, or an iconified ViNCEd window does not have this structure. This means that, as soon as this information is requested, a window must be opened.

That wouldn't be a problem by itself, but there is NO documented way of returning this information which means that it is NOT clear whether this structure is needed any more or not. Since it is quite dangerous to release a resource a program might still need, but there is no standard way of releasing this resource at all, a ViNCEd stream from which this information was requested may neither get iconified nor closed in any other way except for shutting down the stream completely.

ViNCEd, however, introduces a way of giving this resource back - check the DosPackets section in case you're a program author and going to implement something like this.

This argument uses this documented way of giving the resource back, but does not ensure that it is, indeed, no longer required. That is, unfortunately, your job. Use it wisely! Calling "SetVNC FreePointer", or even worse, "SetVNC FreePointer All" at a time WHERE this intuition window structure IS still required, and then iconifying the window will definitely crash the system. There is, unfortunately, no standard way of finding out whether "FreePointer" is safe or not.

1.256 SETCONSOLE/S

This argument is rarely used, and should be avoided. It is definitely for "experts only". What it does is that it sets the "Console Task" pointer of the calling shell to the console controlling the output stream of the SetVNC program on invocation.

This effectively changes the owner of the task using this console pointer. However, this is rarely required.

1.257 LOAD/K (PREFS/S GLOBAL/S WINDOW/S DEFAULT/S LAST/S FROM) (SetVNC Argument)

This loads the ViNCEd preferences from a specified position, usually to edit them with the SetVNC user interface or to install them in the window SetVNC was called from. The following parameters are available:

PREFS/S:

Loads the globally active preferences.

GLOBAL/S:

Identically to the keyword "PREFS".

WINDOW/S:

Loads the preferences used in the window SetVNC was invoked from. This is the default location where SetVNC reads the preferences from if they are required, for example for the GUI, and no other LOAD option was specified.

DEFAULT/S:

Loads the SetVNC "factory default" settings.

LAST/S:

Loads the "Last Saved" preferences, i.e. those that have been saved back to disk. This is not necessarily identical to "GLOBAL", which represent the settings that have been made active by the "Use" button of the SetVNC GUI alone.

FROM:

An parameter requiring an additional argument, namely the file name of a preferences file previously saved with SetVNC. The preferences data is then loaded from the given file. The "FROM" argument itself can be dropped if a file name is provided.

1.258 PREFS/S (SetVNC argument)

This switch disables the "To Window" button in the SetVNC user interface and changes the default operation of the LOAD and SAVE keywords to load or save the global rather than the local window preferences. Thus, it turns SetVNC into a preferences editor for the global ViNCEd settings rather than for the local settings. Usually not required unless you really want to modify the global preferences.

This flag is, for example, used by the "SetVNC" icon in the Prefs drawer of your system.

1.259 PUT/K (SCREEN HISTORY) (SetVNC argument)

This keyword saves the review buffer or the command history to a text file.

PUT SCREEN

this takes an additional parameter, namely a file name to save the screen contents as. The output file will be a text file with some embedded CSI sequences that describe the style and type of the text in the display buffers , as for example whether it was typed in as input or printed output. This file can be later on reloaded with the GET argument and will be placed into the display buffer. You could, in principle, simply copy it to the console using either the "type" or "copy" command, but the direct SetVNC approach is not only faster, but also restores user input as user input, which a simple copy operation is not able to do.

PUT HISTORY

takes an additional parameter, the file name to save the history contents under. The output file will be a plain ASCII file. It can be restored later on with the GET argument.

This argument is, for example, used by the history shell script to prompt the history contents on the screen.

1.260 GET/K (SCREEN HISTORY) (SetVNC argument)

This keyword loads the review buffer or the command history from a text file.

GET SCREEN

this takes a file name as an additional parameter, the file to load the review buffer from. The file might include a subset of known control sequences that describe the style and the type of the text, i.e. whether it was printed or typed input. Note that this attribute cannot be restored simply by "type"ing or "copy"ing the text file to the screen, but note further that only a small subset of the CSI sequences usually known are valid here, mostly for speed reasons.

GET HISTORY

loads the command history from a given text file and replaces the currently active history completely.

1.261 RESET/S (SetVNC argument)

If this argument switch is present, SetVNC will reset the terminal emulation on exit, using the "Esc c" escape sequence .

1.262 MONITOR/A (SetVNC argument)

This argument is similar to the MONITORID argument in the sense that it sets the default monitor which ViNCEd should use to open on its own screen. The difference is that the argument to "MONITOR" must be the name of the monitor as it shows up in the monitor data base, and not the ID of the monitor.

The monitor database can be browsed, for example, by the "ScreenMode" preference editor.

Be warned: Monitor names depend on the localization, hence do not use this argument in script files that should run on Amigas localized for a different language.

Examples:

SetVNC Monitor "NTSC:High Res"

sets the default view mode for screens to be used by ViNCEd to NTSC high res. Please note the double quotes required to enclose the white space of the argument.

1.263 MODIFY/S (SetVNC argument)

This argument must be given to run the graphical preferences editor of SetVNC. This "MODIFY" is implicit if no argument is given at all, but required to launch the editor in all other cases.

For example, the following line would load the preferences from a file named "foo", and display them in the SetVNC editor:

SetVNC LOAD foo MODIFY

1.264 SAVE/K (NOICONS/S PREFS/S GLOBAL/S WINDOW/S TO) (SetVNC argument)

This argument together with its parameters will save or install the preferences in various destinations. The preferences that will be saved are either the preferences of the window SetVNC was invoked from, or any other source specified with the LOAD argument, possibly modified with the preferences editor if the MODIFY switch was used.

NOICONS

This is simply a parameter to "SAVE" to tell SetVNC not to save an icon along with the preferences file.

PREFS

Saves the preferences at their default location in ENV:ViNCEd.prefs and ENVARC:ViNCEd.prefs and activates them such they will be used by all windows that will open afterwards. This does not change the configuration of any window already open. This is identical to what the "Save" button in the preferences editor does except that it does not change the configuration of the window SetVNC was invoked from.

GLOBAL

Makes the preferences system wide available and saves them to ENV:ViNCEd.prefs, but does not save them permanently to disk. This command does not change the configuration of windows already open, it is identical to the "Use" button of the preferences editor except that it does not change the configuration of the active window.

WINDOW

Installs the preferences in the window SetVNC was invoked from. Does not change the configuration of other windows, neither the currently active configuration nor that on disk.

TO

This parameter takes an additional sub argument, the name of a file to save the preferences in. The settings saved in this file can be restored later on with the LOAD argument. This parameter does not install the preferences in any window nor are the preferences made available for other windows. If a file name is present, the keyword "TO" can be omitted completely.

1.265 IFVNC/S (SetVNC argument)

This argument checks whether SetVNC was run from a ViNCEd shell or not. The return code 5 is set if not, otherwise a 0 is returned. This return code can be used in script files, using a following "if" instruction to check whether the script was run from a ViNCEd shell.

To give an example how this could be used:

SetVNC IfVNC if WARN echo "Ugly old console running" else echo "Yeah, ViNCEd active" endif

1.266 Rebuild Delay/REBUILDDELAY (Prefs Flag)

If this flag is enabled, ViNCEd is allowed to delay the window refresh to allow faster printing. The output might jump-scroll if ViNCEd considers that standard scrolling is too slow.

The timing constants for the scrolling are found on the timing page of the SetVNC program .

Vertical scroll threshold sets the minimal tolerated scroll delay. If scrolling is slower than this value, ViNCEd will enable jump scrolling.

Rebuild Delay sets the jump scroll interval. After detecting a slow output, ViNCEd will delay further output by this time interval, collecting all incoming data and printing them at once when this delay is over.

1.267 Unrestricted Cursor Movement/DOSCURSOR (Prefs Flag)

If this flag is enabled, all user cursor movement functions will ignore the difference between user input and printed output and will act as if it sees only user input. The only keyboard function that does still sees a difference is the Send Inputs keyboard function.

1.268 Standard CR Insertion at Start of Line/CRINSERT (Prefs Flag)

If the Send Inputs keyboard function or a related function is invoked with the cursor placed at the beginning of a line, a blank line will be inserted above the current line; the standard operation which is used in all other cases inserts a line below the current line.

If this flag is set, only the standard method will be used, regardless where the cursor is placed.

1.269 Overwrite Mode/OVERWRITE (Prefs Flag)

If this flag is set, the overwrite mode will be used by default. Your inputs will write on top of already printed outputs instead of getting inserted.

This mode can be toggled with the Toggle Overwrite as well; it does not change the behavour of printed output, only user inputs are affected.

1.270 History Buffer Wraps Around/WRAP (Prefs Flag)

If this flag is set, the command history will "wrap around" if its end is reached using the keyboard History Up and History Down functions.

Additionally, the TAB Expansion list will wrap around, too.

1.271 Call Macro to Close Window/SMARTCLOSE (Prefs Flag)

If this flag is enabled, and the window is in Shell Mode , ViNCEd will invoke a macro to close the current window - which might for example run the "EndCLI" command. In all other cases, an "End Of File" condition is generated.

Which macro is run depends on which program ViNCEd detects in its window. If it is a standard shell, the Quit Shell macro will be used, in all other cases the Quit Program . If the according macro is left empty, ViNCEd will also generate an EOF condition.

1.272 Don't send EOF until everybody waits/CLOSEQUEUE (Prefs Flag)

This is a very technical flag and it is usually not worth playing with it. It changes the behavour of ViNCEd if more than one process is running in a window with the AUTO window path argument set.

If this flag is unset, and one of the processes waits for inputs, ViNCEd will generate an EOF condition or invoke a macro to shut down the process.

If this flag is set, ViNCEd will just close the window to wait for the other processes and will only generate the EOF as soon as all processes wait.

There's usually not much sense in setting this flag, just leave it alone.

1.273 Don't Write Printed Text into Clipboard/CUTUSER (Prefs Flag)

If a block is marked with the mouse and later on copied to the clipboard, probably using the edit menu , all characters usually go into the clipboard, regardless whether they are printed or inputs.

If this flag is set, only user input will be kept; printed characters are ignored. This has the advantage that, when the clipboard is later on reinserted, the printed output can't be mis-interpreted as commands.

1.274 Chunky Pixel Graphics/CHUNKYPIXEL (Prefs Flag)

ViNCEd is usually smart enough whether the native graphics organization or the "chunky" organization is used; if it detects a native display, smarter output routines are used that try to speed up scrolling if possible. However, if erroneously a native display is detected where a non-native display is used, this mechanism may slow down scrolling dramatically, or might even create artifacts. If you run into trouble with scrolling on your gfx-board, turn on this flag.

It is also possible to specify the type of graphics on the window open time by the "CHUNKY" and "PLANAR" window path arguments . The first one selects "chunky graphics" like a checked gadget would, and the second switches back to the default, Amiga type graphics organization.

REMARK: ViNCEd does not crash if the type does not match, it just gets slow; Unless, of course, your graphics card software is not compatible to Commodore's programming guide lines.

1.275 Use Shell Mode by Default/SHELLMODE (Prefs Flag)

Selects whether ViNCEd should default to Shell Mode even though no SHELL argument is given in the window path . If this flag is set, the shell mode must be turned off explicitly with NOSHELL .

1.276 Auto Indent Mode/AUTOINDENT (Prefs Flag)

If this flag is set, the Send Inputs keyboard function and related functions won't place the cursor in the first row of a new line, but will however insert as many spaces as needed to align the input to the characters in the line above.

This mode is incompatible to shell usage, don't set it unless you know what you're doing.

1.277 Don’t Add Close Gadget by Default/NODEFAULTCLOSE (Prefs Flag)

If this flag is set, ViNCEd will not add a close gadget by default, but requires the CLOSE window path argument to do so.

If this flag is not set, a close gadget will be added to all ViNCEd windows unless it is turned off with NOCLOSE .

1.278 Implicit Copy after Text Marking/AUTOCOPY (Prefs Flag)

If this flag is selected, ViNCEd will copy the marked block immediately after you release the mouse button and won’t wait for the Copy function. This makes ViNCEd sort of compatible to the "XTerm" unix program.

1.279 Prevent Accidental Window Closing/SAFERCLOSE (Prefs Flag)

If this flag is set, ViNCEd runs in the Shell Mode and more than one process uses the window to be closed, ViNCEd will disable the close gadget which will only shut down one process in this window at a time.

Additionally, a close requester may be enabled.

1.280 Don’t Add Iconify Gadget by Default/FORBIDICONIFY (Prefs Flag)

If set, ViNCEd will not add an iconification gadget to the window unless you ask for it explicitly using the ICONIFY argument of the window path .

If this flag is not set, ViNCEd will usually provide an iconification gadget; you’ve to use the NOICONIFY option to disable it explicitly in this case.

1.281 Disable Middle Mouse Button/DISABLEMMB (Prefs Flag)

If selected, ViNCEd will disable the function of the middle mouse button to insert the text in the clipboard at the location pointed to by the mouse. It will be available for other applications again.

However, even with the disabled middle mouse button, the replacement sequence Ctrl LeftMouse will still function.

1.282 Enable Close Requester/CLOSEREQ (Prefs Flag)

If this flag is enabled, and the Prevent Accidental Window Closing is enabled, too, ViNCEd will open a safety requester as soon as you try to close a window with more than one process running in it.

This might prevent certain surprises when shutting down the shell in a window which is needed by other processes as well.

1.283 Keep Bottom of Window Adjusted/SCROLLTOBOTTOM (Prefs Flag)

If a ViNCEd window gets enlarged, ViNCEd will usually insert blank spaces at the bottom of the window to avoid moving the cursor. If this flag is set, however, the text will be scrolled downwards to align it to the bottom border.

1.284 Inhibit Horiz. Scrolling by DOS Output/NOPRINTSCROLL (Prefs Flag)

If this flag is set, ViNCEd will not scroll the display horizontally if the cursor gets out of range by printing characters. The window will be scrolled "in place" as soon as you start typing.

This mode might be very confusing since the cursor might become invisible from time to time. However, it avoids unnecessary flicker when printing over sized lines.

This flag is also controlable by the CSI sequences CSI >?18l and CSI >?18h.

1.285 Line Break at Right Border/SMALLDISPLAY (Prefs Flag)

If this flag is enabled, the output will be split at the right border of the window, not at the right end of the logical ViNCEd line (which is approximately 240 characters long).

This splitting is permanent, ViNCEd does not reformat paragraphs if the window gets resized, nor does it reformat the output in any way. This behavour is also compatible to what XTerm does.

This flag is also available in the form of a CSI sequence , namely CSI >?19l and CSI >?19h.

1.286 Destructive DEL and BS/DOSERASE (Prefs Flag)

If this flag is set, the control characters DEL 0x7f (delete) and BS 0x08 (backspace) are enabled and destructive, i.e. will remove characters from the screen.

If the flag is disabled, the BS sequence will move the cursor backwards, but won't erase characters. The DEL sequence will print a "smear" character and won't be regarded as control sequence. Most programs won't expect this when printing a backspace control sequence, so I advice you not to turn on this option.

The CSI sequences CSI >?24h and CSI >?24l can be used to set this flag "on line".

If this flag is set, you should set the Insertion mode for DOS output as well.

Warning: This flag is incompatible for shell usage. Don't set it unless you know what you're doing.

1.287 Notify DOS About Paste/AUTOPASTE (Prefs Flag)

If this flag is set, the Paste menu item will not insert the clipboard contents, but will instead sent the CSI sequence CSI 0 v. It is then up to the listening program to handle the clipboard insertion by itself. However, regardless of this flag, Cut, Copy and all the remaining block operations will work as normal, but the middle mouse button is also affected, since it inserts the clipboard, too.

This flag is incompatible to shell usage, don't set it unless you know what you're doing.

However, a program might choose to disable temporary the automatic paste by ViNCEd. This is done by the CSI sequence "CSI >?25l". Use "CSI >?25h" to turn this on.

REMARKS: The paste behavour of ViNCEd does NOT change if a window is in raw mode instead of cooked mode , like for the standard console handler.

There is a second flag related to the one presented above, which can't be set by default because it is even more restrictive. If you send the CSI sequence "CSI >?27h" to ViNCEd, all the block operations, including Cut and Copy and the selection of blocks must be done by you. Special CSI sequences will be send to indicate a Cut or Copy request, and the block marking is totally your own business, i.e. you have to listen to the mouse movement and -keys. Send a "CSI >?27l" to restore the normal operation.

The command line option "AUTOPASTE" is the same flag, but reversed. "Off" means that ViNCEd DOES NOT insert the clipboard automatically.

Just as a side remark: ViNCEd does not require the "ConClip" program. If you run ViNCEd to replace CON: completely, you may remove "ConClip" from your startup-sequence.

1.288 DISABLESCROLL (Prefs Flag)

If this flag is set to "on", ViNCEd will disable the window scrolling if the bottom border is reached. This preferences flag functions identically to the CSI >1l and CSI >1h control sequences .

This flag is incompatible to shell usage. Don't set it unless you know what you're doing.

1.289 DISABLEWRAP (Prefs Flag)

If this flag is set to "on", ViNCEd will disable automatic line break at the right edge of a line. This is either the right window border if the SMALLDISPLAY flag is set, or the end of the logical line - which is approximately 240 characters long. This flag functions identically to the CSI sequences CSI ?7l and CSI ?7h.

This flag is incompatible to shell usage. Don't set it unless you know what you're doing.

1.290 Insertion Mode for DOS Output/DOSINSERT (Prefs Flag)

If this flag is enabled, printed characters will get inserted into the already existent output instead of overwriting them. This flag DOES NOT change the keyboard insertion or overwrite mode, which is controlled by a different flag .

Most software does not expect insertion mode, so leave this flag off if you don't have a good reason not to do.

The CSI sequences CSI 4l and CSI 4h will clear and set this flag "on line".

If this flag is set, you should also set the Destructive DEL and BS flag.

This flag is incompatible to shell usage. Don't set it unless you know what you're doing.

1.291 VT-220 Compatibility Mode/VTMODE (Prefs Flag)

If this flag is set, ViNCEd will run a VT-220 terminal emulation instead of the usual CON: emulation; the meaning of some control sequences will therefore change.

This flag is incompatible to shell usage; don't set it unless you know what you're doing.

The CSI sequence CSI >?2h will also enable this emulation, CSI >?2l will disable it.

1.292 ROWLOCK (Prefs Flag)

If this flag is enabled, ViNCEd won't allow the user to leave the current input line by any means. Even Return won't leave the current input line, the vertical cursor movement functions are disabled completely, and some other functions get restricted, too. The main use of this flag is to provide a mechanism to use ViNCEd as a terminal for a "mask input" such that the user can't overwrite the mask.

This flag is also available in the form of the CSI >?3h CSI sequence , CSI >?3l will disable it again.

This flag is definitely NOT for shell usage. Don't touch it unless you know what you're doing.

1.293 Underline Cursor/UNDERLINE (Prefs Flag)

If this flag is set, ViNCEd will use an underline cursor instead of the standard block cursor.

This flag is also available by the CSI sequence CSI >?4h. CSI >?4l will restore the standard cursor shape.

1.294 Blinking Cursor/BLINKING (Prefs Flag)

If this flag is set, ViNCEd will blink the cursor. The cursor blink speed can be setup on the timing page of SetVNC .

This flag can be set by the CSI sequence CSI >?6h as well.

1.295 XTerm/CON: Cursor Mode/XTERMCURSOR (Prefs Flag)

If this flag is enabled, ViNCEd won't move the cursor if the window contents are scrolled with the slider gadgets . Instead, the cursor will be scrolled out of the window. The window will "snap back" as soon as you press another key.

This flag can be set with the CSI sequence CSI >?30h as well.

1.296 Disable BS at Start of Line/NOBSSTART (Prefs Flag)

By setting this flag, you may disable the backspace keyboard function (or, to be fully correct, the "Delete Backwards" function) at the beginning of a line - where it would usually erase the current line and join it to the previous line.

1.297 ANSI Colors as Default/ANSIMODE (Prefs Flag)

If this flag is enabled, ViNCEd will use the ANSI coloring scheme by default, as if ANSI was given in the window path; the window path argument NOANSI must be given explicitly to use the standard CON: coloring again.

1.298 Inverse ANSI Coloring/ANSIREVERSE (Prefs Flag)

This flag will swap the ANSI colors 0 and 7, i.e. black and white. Hence, the default ANSI coloring will be black text on white background instead of white text on black background.

If the ANSI mode is not enabled, this flag will do nothing.

1.299 Numeric keypad cursor control/NUMPADMODE (Prefs Flag)

This flags sets the "NumLock" keyboard qualifier by default and makes therefore certain keyboard functions available; these are usually cursor movement functions on the numeric keypad.

The "NumLock" qualifier is a keyboard qualifier much like "CapsLock", except there is no LED available to display its state. It can be toggled "by hand", too, by binding the "Toggle NumLock" keyboard function to an appropriate key - this is Alt Num[in the default configuration.

1.300 Extended Colors Instead of Bold/BOLDEXTCOLORS (Pref Flag)

If this flag is set, ViNCEd will interpret the "Bold" text style in a different way. Instead of printing bold characters in boldface, ViNCEd will shift to the extended color map and use the colors 8 to 15 instead of the colors 0 to 7 for the text. The "blinking" attribute will, in a similar way, shift the colors of the text container by eight to use the extended color set - instead of getting ignored; the current ViNCEd version does not yet support "blinking" characters, but already keeps the "blinking" attribute.

1.301 No Insertion Into Border/SHORTSCREENINSERT (Prefs Flag)

If enabled, this flag changes the operation of the CSI L and CSI M control sequences slightly; instead of inserting and removing lines from the complete lower display buffer , these sequences will ignore lines beyond the bottom window edge complete and will leave them alone.

This flag is mainly a compatibility kludge for programs that can't handle with the complete ViNCEd review buffer, as for example "vim".

1.302 Don't scroll into the border (Prefs Flag)

If this flag is enabled, you won't be able to extend the review buffer by scrolling into blank unoccupied areas. The cursor will just stop at the upper or lower end of the review buffer .

This flag won't affect output, only user cursor movements are restricted.

1.303 Invisible Type Ahead/TYPEAHEAD (Prefs Flag)

If this flag is enabled and there is currently no program waiting for your input, the inputs will be kept invisible in an internal buffer and will be printed as soon as they are requested.

Note: Not all characters are kept in this type ahead buffer, some are "immediate", as for example the keyboard functions Suspend and Send ^C which have to work all the time for obvious reasons.

1.304 Disable Horizontal Scroller/DISABLEPROPX (Prefs Flag)

This flag disables adding the horizontal scroller to ViNCEd windows unless the window path argument PROPX re-enables it explicitly again.

1.305 Disable Horizontal Scroller/DISABLEPROPY (Prefs Flag)

This flag disables adding the vertical scroller to ViNCEd windows; to get a vertical scroller then, it must be requested explicitly with the window path argument PROPX .

1.306 Enable Scrollers in RAW Mode/RAWSCROLLERS (Prefs Flag)

If the window is switched to the RAW console mode (also called the "unprocessed" or "non-canonical" mode which disables the ViNCEd editor functions), all scrolling is usually forbidden; this includes that the horizontal and vertical scrollers will be ghosted and are no longer available. Another side effect is that you won't be able to scroll the window contents implicitly my marking a block and moving the mouse pointer to the edge of the window.

If this flag is enabled, the scrollers will stay available in all console modes; block marking will also work as usual, including scrolling the window contents if necessary.

1.307 Rigid XTerm Cursor/RIGIDCURSOR (Prefs Flag)

If this flag is set, you won't be able to set the text cursor by "clicking" at a new position with the mouse pointer. This flag restricts therefore the XTerm mode even more and makes all cursor movement impossible, except using the keyboard.

If you even don't want to move the cursor by the keyboard, then change the keyboard definition to change the keyboard functions bound to the cursor keys.

1.308 Keep Duplicates in the History/KEEPDOUBLES (Prefs Flag)

If you enter a command, ViNCEd will rewind the command history and add your command line to the end of it, except this command line is already present at the end; Hence, ViNCEd will not keep duplicate entries, i.e. repeated commands, in its history.

However, by setting this flag, ViNCEd can be told to keep even these lines, regardless whether it is an repeated command or not.

1.309 Do not Place File Requester/DONTPLACEREQUESTER (Prefs Flag)

If this flag is set, ViNCEd does not attempt to place the TAB expansion file requester using the position information on the sixth shell page of the SetVNC program. Instead, placement is left to the requester package.

This is mainly a compatibility kludge for the "reqchange" packet which interprets the requester coordinates of the ASL tags as a position relative to the parent window instead using absolute coordinates.

1.310 History lines/BUFFERSIZE (Prefs Setting)

This is the size of the command history in lines, and integer value between 5 and 4096.

1.311 Upper display size/UPPERSIZE (Prefs Setting)

The number of lines kept in the upper review buffer that are scrolled out of the top edge of the window. If more lines are scrolled into that window than the buffer can hold, the upmost line is deleted.

The contents of this buffer is not visible, but it holds lines that can be scrolled back into the window, for example using the scroller gadgets .

This buffer must be between 64 and 4096 lines large.

1.312 Lower display size/LOWERSIZE (Prefs Setting)

The number of lines kept in the lower display buffer . This includes the lines kept in the window itself, and the lines scrolled out of the lower edge of the window; this happens for example if the window contents is scrolled downwards to make the upper review buffer visible.

If more lines are scrolled into this window, the bottommost line of this buffer is deleted.

This buffer size must lie between 128 and 4096 lines; it should be at least as large as the maximal possible window size in lines, plus the size of the upper display buffer .

1.313 Cached directories/CACHESIZE (Prefs Setting)

This is the size of the TAB expansion cache, in directories. If the buffer overflows during an expansion, the latest directory visited while expanding will be flushed.

The TAB expansion cache must be one to 256 directories large.

1.314 Intuition Delay/DELAYINTERVAL (Prefs Setting)

This setting is a mess, and it's only needed if you work with the old 1.3 workbench (why ?). Due to a bug in the old intuition.library, after resizing or moving a window the calling program must wait some time before the window parameters get updated and the refresh statement is set. Even worse, the whole refresh timing is a mess. ViNCEd tries to avoid such problems by waiting this time, given in milliseconds (thousands of a second) until intuition performs the necessary operations. This time is somewhat processor dependent, and you might try to choose a smaller value if you have a faster computer.

There is no need to touch this if you use the workbench 2.04 or up, since this timing is no more needed then.

In all other cases, a number between 50 and 1000 is required.

1.315 Rebuild Delay/REBUILDTIME (Prefs Setting)

After text output has been delayed by an enabled REBUILDDELAY flag, either because it is slower than the speed specified by the vertical scroll threshold , or a horizontal scroll is necessary, this is the time in milliseconds (thousands of a second) ViNCEd buffers all incoming text and suppresses output. The complete text is printed when the time entered in this field is over. The timing of the scrollers on the edges of the window is controlled by this value, too, because scroller refresh is relatively slow.

Use values from 25 (no remarkable delay) to 1000 (quite a lot of time passes until refresh), but remember:

If you choose a large value here, a lot of text might get printed, and it might get scrolled out of the window before you see it!

Also, I advise you to select AT LEAST twice the value of the Vertical scroll threshold , or timing might get a mess: Delaying the output using these settings won't speed up anything considerably.

1.316 Vertical scroll threshold/SCROLLTIME (Prefs Setting)

This setting controls the delayed window refresh feature of ViNCEd, if enabled. If the graphics output, for example scrolling, is too slow, ViNCEd does not attempt to output text immediately but keeps it in an incoming buffer.

This setting specifies what "too slow graphics output" means in detail. If text can't get printed by a faster rate than this, given in milliseconds (thousands of a second),is delayed and will be printed when there is enough time or at least after the rebuild delay .

Admissible values range from 5 (things are really in a hurry, MOVE ON!) to 500 (let's take things easy). If you have a slow processor (like the MC68000) or slow graphics (many colors), be a bit more tolerant and increase this value!

REMEMBER: "Slow" for a MC68060 is something different than for a MC68000!

1.317 Cursor Blink Speed/BLINKSPEED (Prefs Setting)

This selects the blink speed of the cursor, if this feature is enabled.

Values are given in milliseconds (thousands of a second) and range from 100 (very annoying flickering) to 1000 (boredom).

1.318 Monitor/MONITORID (Prefs Setting)

Selects the default monitor ID when ViNCEd opens its own screen, controlled by other fields of the window path , i.e. the hardware details of the screen where the window should appear on.

The SetVNC program offers, too, a screen mode requester for convenience to setup this value, on the first window page . However, Os 2.1 or better is required for this requester.

Hex notation using "0x" or "$" is valid here, or should be even preferred, but the upper sixteen bits (the "monitor ID" itself) are ignored if you run ViNCEd on a 1.3 workbench.

1.319 Path Only Qualifier/PATHONLYQU (Prefs Setting)

This is the setup for the "path only" keyboard qualifier of the icon drop function of ViNCEd. If you hold the qualifiers given here, ViNCEd will only insert the path of the object dropped on its window.

Note: The icon drop function will not distinguish between the left and right qualifier keys, e.g. left and right shift will work alike. It will require the shell mode , too.

1.320 Name Only Qualifier/NAMEONLYQU (Prefs Setting)

This is the setup for the "name only" keyboard qualifier of the icon drop function of ViNCEd. If you hold the qualifiers given here, ViNCEd will only insert the name of the object dropped on its window and won't insert the complete path of the icon.

Note: The icon drop function will not distinguish between the left and right qualifier keys, e.g. left and right shift will work alike. It will require the shell mode , too.

1.321 Cursor Color Setup/CURSOR COLOR (Prefs Setting)

ViNCEd can be told to use a cursor of a specific color instead just inverting the color on the screen. For that, load the SetVNC program and go to the second window page . The "« Prev" and "Next »" buttons at the bottom of the window select the color register to adjust, with the cursor color being the "previous" color of color register 0, i.e. click again on the "« Prev" gadget when the text above says that color 0 is currently shown.

To activate the colored cursor, enable the "Load register" checkmark and select your preferred color with the sliders above.

The "Define ANSI mapping" gadget below has a strange meaning for the cursor color - since there is actually no ANSI rule what the cursor color should be. If you activate this gadget as well, ViNCEd will allocate the color for the cursor with a higher priority which is especially useful on screens offering few free pens.

In the preferences file the cursor color is specified with the keyword "CURSORCOLOR"; this keyword takes five arguments, separated by commas, reflecting the state of the "Load register" and "Define ANSI mapping" gadgets, as well as the states of the scrollers defining the color.

The first argument is either LOAD or NOLOAD, the second ANSI or NOANSI. These are the checkmark gadget states. The remaining three arguments specify the cursor color red, green and blue components, respectively, as sixteen bit fractions in hex notations; i.e. 0x0000 is minimal intensity, 0xffff is maximal intensity and 0x7fff is, for example, half intensity.

1.322 Color Setup/COLOR (Prefs Setting)

This defines one of the sixteen ViNCEd colors; the color to be defined can be selected with the "« Prev" and "Next »" gadgets on the second window page of SetVNC , they decrease or increase the color index shown above on the page; the color "previous" to color 0 is, by the way, the cursor color .

A color register can either specify a hardware register that should be loaded by ViNCEd with a specific color if ViNCEd opens on its own screen, or defines the color of one of the ANSI registers if that feature is used.

To load a hardware register, check the "Load Register" gadget on the left side of the page; this will make sense only if ViNCEd opens on its own screen because ViNCEd won't load any hardware register just for a window.

Remember, however, that the first four pens are usually reserved for the system to draw the gadget and window artworks, hence loading these registers might result in weird looking imaginary and probably the loss of the "3D style"; to compensate, select a dark color for the register #1, and a light color for register #2.

Note: Even non-reserved colors will be used by ViNCEd, it just doesn't load the hardware register on startup. However, WHICH colors will appear in the non-ANSI coloring mode is then up to the system and on the system wide "Palette" defaults. You may even load the hardware registers explicitly with the "CSI V" control sequence "on line", but the registers defined by the ViNCEd preferences will be reloaded on a reset sequence Esc c.

To define an ANSI color that overrides the ANSI color defaults , check the second "Define ANSI mapping" gadget. ViNCEd will then try to allocate a shared pen for rendering characters of the ANSI pen value defined; this will work even for windows opened on a public screen, but makes obviously only sense if the "ANSI" coloring is enabled - either by defining this as a default , or by specifying the "ANSI" argument as part of the window path .

However, if only few free pens are available on that screen, ViNCEd may fail to allocate all required pens and the colors may or may not look like what you expect.

It makes sense to set both flags. ViNCEd tries in this case first to arbitrate the selected hardware register to load the desired color for the ANSI pen of the same index. If this register is not available, ViNCEd tries to allocate a shared pen for the color, but with a higher precision than with only the ANSI flag selected.

The CSI V sequence does, for example, reserve ANSI colors by setting both flags.

The preferences file defines the sixteen color registers using the "COLOR" keyword; the register loaded is implicit in the order in which these keywords appear - the first "COLOR" keyword loads register #0, the next register #1, etc. Each keyword takes five arguments, to be separated by commas. The first two arguments reflect the state of the two checkmark gadgets discussed above, and are either LOAD/NOLOAD or ANSI/NOANSI respectively. The next three arguments represent the red, green and blue components of the color to be loaded, as sixteen bit fractional values in hexadecimal notation, i.e. 0x0000 is the lowest possible intensity, 0xffff is full intensity.

1.323 TAB_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the first TAB expansion

keyboard function .

1.324 TAB_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the first TAB expansion

keyboard function .

1.325 TAB_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the first TAB expansion

keyboard function .

1.326 TAB_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the first TAB expansion keyboard function .

1.327 TAB_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the first TAB expansion keyboard function .

1.328 TAB_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the first TAB expansion keyboard function .

1.329 TAB_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the first TAB expansion keyboard function .

1.330 TAB_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the first TAB expansion keyboard function .

1.331 TAB_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the first TAB expansion keyboard function .

1.332 TAB_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the first TAB expansion keyboard function .

1.333 TAB_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the first TAB expansion keyboard function .

1.334 TAB_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the first TAB expansion keyboard function .

1.335 TAB_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the first TAB expansion keyboard function .

1.336 TAB_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the first TAB expansion keyboard function .

1.337 TAB_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the first TAB expansion
keyboard function .

1.338 SRT_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the second TAB expansion
keyboard function .

1.339 SRT_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the second TAB expansion
keyboard function .

1.340 SRT_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the second TAB expansion
keyboard function .

1.341 SRT_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the second TAB expansion keyboard function .

1.342 SRT_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the second TAB expansion keyboard function .

1.343 SRT_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the second TAB expansion keyboard function .

1.344 SRT_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the second TAB expansion keyboard function .

1.345 SRT_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the second TAB expansion
keyboard function .

1.346 SRT_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the second TAB expansion keyboard function .

1.347 SRT_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the second TAB expansion keyboard function .

1.348 SRT_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the second TAB expansion keyboard function .

1.349 SRT_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the second TAB expansion keyboard function .

1.350 SRT_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the second TAB expansion keyboard function .

1.351 SRT_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the second TAB expansion keyboard function .

1.352 SRT_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the second TAB expansion keyboard function .

1.353 DEV_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the third TAB expansion keyboard function .

1.354 DEV_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the third TAB expansion keyboard function .

1.355 DEV_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the third TAB expansion keyboard function .

1.356 DEV_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the third TAB expansion keyboard function .

1.357 DEV_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the third TAB expansion keyboard function .

1.358 DEV_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the third TAB expansion keyboard function .

1.359 DEV_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the third TAB expansion keyboard function .

1.360 DEV_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the third TAB expansion keyboard function .

1.361 DEV_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the third TAB expansion keyboard function .

1.362 DEV_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the third TAB expansion keyboard function .

1.363 DEV_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the third TAB expansion keyboard function .

1.364 DEV_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the third TAB expansion keyboard function .

1.365 DEV_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the third TAB expansion keyboard function .

1.366 DEV_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the third TAB expansion keyboard function .

1.367 DEV_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the third TAB expansion keyboard function .

1.368 DIR_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the fourth TAB expansion keyboard function .

1.369 DIR_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the fourth TAB expansion keyboard function .

1.370 DIR_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the fourth TAB expansion keyboard function .

1.371 DIR_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the fourth TAB expansion keyboard function .

1.372 DIR_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the fourth TAB expansion keyboard function .

1.373 DIR_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the fourth TAB expansion keyboard function .

1.374 DIR_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the fourth TAB expansion keyboard function .

1.375 DIR_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the fourth TAB expansion
keyboard function .

1.376 DIR_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the fourth TAB expansion
keyboard function .

1.377 DIR_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the fourth TAB expansion
keyboard function .

1.378 DIR_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the fourth TAB expansion
keyboard function .

1.379 DIR_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the fourth TAB expansion
keyboard function .

1.380 DIR_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the fourth TAB expansion
keyboard function .

1.381 DIR_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the fourth TAB expansion
keyboard function .

1.382 DIR_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the fourth TAB expansion
keyboard function .

1.383 INF_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the fifth TAB expansion
keyboard function .

1.384 INF_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the fifth TAB expansion
keyboard function .

1.385 INF_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the fifth TAB expansion
keyboard function .

1.386 INF_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the fifth TAB expansion keyboard function .

1.387 INF_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the fifth TAB expansion keyboard function .

1.388 INF_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the fifth TAB expansion keyboard function .

1.389 INF_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the fifth TAB expansion keyboard function .

1.390 INF_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the fifth TAB expansion keyboard function .

1.391 INF_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the fifth TAB expansion keyboard function .

1.392 INF_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the fifth TAB expansion keyboard function .

1.393 INF_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the fifth TAB expansion keyboard function .

1.394 INF_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the fifth TAB expansion keyboard function .

1.395 INF_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the fifth TAB expansion keyboard function .

1.396 INF_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the fifth TAB expansion keyboard function .

1.397 INF_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the fifth TAB expansion keyboard function .

1.398 ALT_FILE_PRI (Prefs Setting)

The expansion priority for ordinary files of the sixth TAB expansion keyboard function .

1.399 ALT_EXEC_PRI (Prefs Setting)

The expansion priority for executables of the sixth TAB expansion keyboard function .

1.400 ALT_SCRIPT_PRI (Prefs Setting)

The expansion priority for script files of the sixth TAB expansion keyboard function .

1.401 ALT_PATH_PRI (Prefs Setting)

The relative expansion priority for files in the command path of the sixth TAB expansion keyboard function .

1.402 ALT_COMMAND_PRI (Prefs Setting)

The relative expansion priority for files in the C: directory of the sixth TAB expansion keyboard function .

1.403 ALT_RESIDENT_PRI (Prefs Setting)

The expansion priority for files in the shell resident list of the sixth TAB expansion keyboard function .

1.404 ALT_INFO_PRI (Prefs Setting)

The expansion priority for icon (.info) files of the sixth TAB expansion keyboard function .

1.405 ALT_DEVICE_PRI (Prefs Setting)

The expansion priority for devices of the sixth TAB expansion keyboard function .

1.406 ALT_ASSIGN_PRI (Prefs Setting)

The expansion priority for assigns of the sixth TAB expansion
keyboard function .

1.407 ALT_VOLUME_PRI (Prefs Setting)

The expansion priority for volume of the sixth TAB expansion
keyboard function .

1.408 ALT_DIRECTORY_PRI (Prefs Setting)

The expansion priority for directories of the sixth TAB expansion
keyboard function .

1.409 ALT_DOUBLEREQ (Prefs Setting)

The Double TAB Requester flag of the sixth TAB expansion
keyboard function .

1.410 ALT_FULLEXPAND (Prefs Setting)

The Expand Fully flag of the sixth TAB expansion
keyboard function .

1.411 ALT_VNCREQUESTER (Prefs Setting)

The Add ViNCEd matches flag of the sixth TAB expansion
keyboard function .

1.412 ALT_AMBIGREQ (Prefs Setting)

The Requester if ambiguous flag of the sixth TAB expansion
keyboard function .

1.413 Left Edge/REQ_LEFTEDGE (Prefs Setting)

This defines the position of the "left edge" of the TAB expansion requester relative to the left edge of the window; it must be an integer between -2048 and 2048 (let's hope video hardware technology doesn't evolve too quickly...).

This setting is only used if the Do not place file requester is NOT set; the only reason why this can be turned off is to work around a rather strange interpretation of requester coordinates of the reqtools library.

1.414 Top Edge/REQ_TOPEDGE (Prefs Setting)

This defines the position of the "top edge" of the TAB expansion requester relative to the top edge of the ViNCEd window.

This setting is only used if the Do not place file requester is NOT set; the only reason why this can be turned off is to work around a rather strange interpretation of requester coordinates of the reqtools library.

1.415 Width/REQ_WIDTH (Prefs Setting)

This defines the width of the TAB expansion requester in pixels; it must be an integer between 0 and 2048.

If the width is zero, or "too small", ViNCEd will pick default values here.

1.416 Height/REQ_HEIGHT (Prefs Setting)

This defines the height of the TAB expansion requester in pixels; it must be an integer between 0 and 2048.

If the height is zero, or "too small", ViNCEd will pick default values here.

1.417 MACRO (Prefs Setting)

This keyword defines the body text of a macro. The macros are just assigned by order, the first MACRO keyword defines the first macro, and so on, up to ten macro definitions are allowed.

The macro body, hence the argument of this keyword, is just a plain string, including possibly special control sequences to be inserted by the macro.

Note especially that the semicolon ";" may not be used as a comment introducer on these lines, it will be part of the macro body if specified. Note, too, that leading and trailing spaces of the macro body will be removed, too. To avoid stripping of spaces, include the macro body in double quotes.

For details, check the macros section .

1.418 SYSTEMMACRO (Prefs Setting)

These macros define the five internal system macros ViNCEd uses.

The macro body, hence the argument of this keyword, is just a plain string, including possibly special control sequences to be inserted by the macro.

Note especially that the semicolon ";" may not be used as a comment introducer on these lines, it will be part of the macro body if specified. Note, too, that leading and trailing spaces of the macro body will be removed, too. To avoid stripping of spaces, include the macro body in double quotes.

For details, check the macros section .

The meaning of the system macros and the times of invocation are:

First system macro: Quit shell

This defines the command that is used to close a shell window, if any. It must be enabled with the Call macro to close window flag. If this flag is disabled, an "End Of File" condition will be generated. It should call "EndCLI", or another script that shuts down the shell.

This macro is not used if a program is running in the window. ViNCEd uses the Quit program macro in that case.

Second system macro: New window

This is the macro invoked if the "New Window" item of the project menu gets selected. This macro should open another ViNCEd window.

Third system macro: Get help

This macro gets invoked if you select "Help..." from the project menu . It runs by default the SetVNC program to load and display this guide.

Fourth system macro: Edit settings

Invoked if you select the "Settings..." item of the project menu . This runs again SetVNC , but without arguments.

Fifth system macro: Run editor

This macro is not yet used. It is reserved to turn the ViNCEd window into an editor and should be invoked if an "Edit..." item in the project menu gets selected. This is, however, not yet implemented.

1.419 Fork new shell/RUN_NEW_SHELL (Prefs Setting)

This is strictly speaking not a macro, at least it is used quite differently. The contents of this macro do not get inserted into the keyboard buffer, the macro body is, instead, executed directly as a command.

This command is run whenever the Ctrl-Z keyboard function is requested to fork a new shell in the current window, and no free shell is available. It should run a new shell by using the "NewShell" command, in the same window.

Because this macro is used in a different way, the interpretation of the control sequences is slightly different, even though they are passed thru the same parser as for macros .

For the ordinary macros, "\r" is the press of the Return key, but since this string is passed directly to the operating system for execution, a "\n" must be used to mark the end of this command and to start the execution.

It should be set to "NewCLI WINDOW=*\n" for almost all reasons, except when a custom shell should be run instead.

If leading or trailing spaces are desired - for whatever reasons - the double quotes are required since they avoid "trimming" of the command; they are superfluous in all other cases.

1.420 Icon Path/ICON_PATH (Prefs Setting)

This is a string that contains the complete path of an icon - a ".info file" - that should be used as "iconification" for a ViNCEd window. If this path is left empty, ViNCEd will present its default icon.

A ".info" MUST NOT be appended, that's done by ViNCEd automatically.

Note (for nick-pickers): Due to the way how ViNCEd parses the prefs file, leading or trailing spaces are not allowed in this file name. Who wants to do that anyways? (-;

1.421 Icon Title/ICON_TITLE (Prefs Setting)

This string is used as underline title of the icon presented for an iconified ViNCEd window. If this string is empty, no the window title will be used instead.

Exactly like the window title, this string might contain control characters using the percent sign "%" which are expanded by ViNCEd at the time the window gets iconified.

Unlike the window title, however, the current Os does not allow to adapt the icon title in case the contents of the control characters change.

This setting might be overridden by a program sending the "ESC] 1;title BEL" escape sequence .

1.422 Quit Program/QUIT_PROGRAM (Prefs Setting)

This defines a system macro that is invoked whenever a ViNCEd window should be closed with an active and running program in it. It replaces the Quit shell in that case.

However, ViNCEd must be told to use macros to close the window - which is done by the Call macro to close window flag. If this flag is not set, ViNCEd will ignore this macro and will just send an "End of File" condition. This will happen, too, if this macro is left blank.

1.423 Font/DEFAULT_FONT (Prefs Setting)

Here you define the default font for ViNCEd windows - only for the windows contents, not for the menu and the screen title - which is used if no font is specified in the window path .

The font specification must be given by the base name of the font without ".font", i.e. "topaz", a dot, and the requested size. Hence, to use the topaz font, size 9 as default font, specify

topaz.9

here.

This font MUST be a fixed-width (non-proportional) font, or the graphics output will look messy.

It might be somewhat simpler to use the font requester of the SetVNC program on the first window page , just press the "Font" gadget on that page.

Note: The XEN font is NOT a fixed width font. Try the "topaz6.8" of the author included in the distribution.

1.424 Default path/DEFAULT_PATH (Prefs Setting)

This string is used as window path if no path was specified when opening a ViNCEd window, i.e. just the blank handler name like "VNC:" was used. The handler name itself MUST NOT be specified in this string.

Except that, it is interpreted and handled just in the same way as a path specified in the traditional way.

Note: Due to a "feature" of the 1.3 Mount command, this is disabled for older Os releases. Opening a window just with the handler name will (and has to!) fail for these releases.

1.425 BUTTONMACRO (Prefs Setting)

This keyword defines the macro body for the ViNCEd title bar buttons. Each button definition must consist of a pair of a BUTTONMACRO and a following BUTTONTITLE keyword; up to ten buttons may be defined, they will be put into the window title bar, from left to right, as many buttons as room is available.

1.426 BUTTONTITLE (Prefs Setting)

This keyword defines the title of a ViNCEd button , to appear in the ViNCEd title bar. This definition MUST follow a BUTTON-MACRO keyword which has already setup the body of the macro to be defined; up to ten buttons can be defined.

They will be installed into the window title in their order of appearance, from left to right, as many as possible.

1.427 File priority (TAB Setting)

The priority assigned to "ordinary files". These are files that have neither the "e" bit nor the "s" bit set, i.e. are neither scripts nor executables. Since the "e" bit is set by default in the Amiga system, very few files will actually match this category.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.428 Executables priority (TAB Setting)

The priority assigned to files with the "e" (executable) bit set. Note that most Amiga files fall into this category and not into Files since this flag is set by default by the AmigaOs.

If both the "s" and the "e" bit is set, the file is a script , not an executable.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.429 Scripts priority (TAB Setting)

The priority assigned to files with the "s" (script) bit set, hence shell scripts, and mostly ARexx interpreter files. If both, the "s" and the "e" bit are set, the file is supposed to be a script, not an executable .

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.430 Path priority, relative (TAB Setting)

This is a relative priority added to the priority to all files found in the command search path of the shell, except for the current directory and the C: assign. ViNCEd will search in the path only if you're expanding the first argument of a shell line - namely, the command. Neither directories nor icons found in the path match. If this is set to -128, the command search path is not searched at all, and if the overall priority of a file falls below -128, it is not added to the expansion list.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.431 Command (C:) priority, relative (TAB Setting)

The relative priority of files found in the C: assign. This priority is added as a modifier to the priority of objects found in the C: directory if the first argument of a command line is expanded. If a different argument gets expanded, the C: directory is excluded automatically. Additionally, neither icons nor directories in C: match anything, they get just ignored. If you don't want ViNCEd to search C:, even for the first argument, then set this to -128. If the overall priority of a file falls below -128, it is not added to the expansion list.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.432 Residents priority (TAB Setting)

The priority for commands found in the shell internal list of resident commands. This list is only searched if the first argument of a command line is to be expanded and left alone else. If this priority is set to -128, the resident list is never searched.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.433 Icon (.info) priority (TAB Setting)

The priority of the ".info" files that contain the workbench icons. To get rid of them completely, assign them a priority of -128.

Note, too, that icons will not be found in any other directory than the current directory, even if the first argument gets expanded, and even if the C: directory and the path is included in the search path.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.434 Devices priority (TAB Setting)

The priority of DOS device drivers such as "df0:", "ser:" or "VNC:". If you don't want to see device drivers in your expansion list, set this to -128.

Note that this DOES NOT include assigns or volumes .

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.435 Assigns priority (TAB Setting)

The priority of assigns such as "DEVS:" or "LIBS:". If you don't want to see assigns in your expansion list, set this to -128.

Note that this DOES NOT include devices or volumes .

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.436 Volumes priority (TAB Setting)

The priority of volume names such as "Work:" or "Empty:", e.g. names of disks, but not the name of the drive. If you don't want to see volume names in your expansion list, set this to -128.

Note that this DOES NOT include assigns or devices .

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.437 Directories priority (TAB Setting)

The priority assigned to ordinary directories. If you don't want to see directories in your TAB expansion, set this to -128. However, directories will still match if the search pattern ends explicitly with a forwards slash "/", regardless of this priority.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this priority?

1.438 Double TAB Requester (TAB Setting)

If this item is checked, the Double TAB requester is enabled. This requester is presented whenever you press TAB a second time after completion of the directory search, within a double-click time delay. This time interval is under control of the "Input" preferences program in your "Prefs" drawer.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this flag?

1.439 First TAB expands fully (TAB Setting)

If checked, ViNCEd will not try to present a refinement in case more than one matching file was found on a TAB expansion. Instead, the first match found will be inserted immediately.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this flag?

1.440 Add ViNCEd matches to the requester (TAB Setting)

If disabled, a standard file requester showing the current directory will be used. If enabled, only matching files will be shown in this requester, but INCLUDING matches from the resident list, the device list and even matches found in other directories, if available.

This requester might look a bit strange sometimes since it may contain double entries. This is, however, normal if, for example, a file was found twice in two different directories, or was found in the command directory and in the list of resident commands. You should be alarmed if that happens because it is not quite clear which copy of the file will be executed by the shell.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this flag?

1.441 Requester if expansion is ambiguous (TAB Setting)

If this item is enabled, ViNCEd will open its file requester in case more than one possible match for your TAB expansion template was found.

To the TAB expansion tutorial To the TAB expansion settings overview

Where to adjust this flag?

1.442 CSI and ESC sequences

ViNCEd recognizes a big bunch of control sequences; mostly introduced by a CSI (0x9b = 155) or ESC (0x1b = 27) character, these do not only change ViNCEd's internal settings, but do also move the cursor, scroll the window, set the color and font of the text and adjust the keyboard.

All control sequences of the original console handler are supported, most VT-220 and some unix "XTerm" sequences have been added to complete the set, even some SGI winterm sequences are present.

The explanation of some concepts, used in the description of the sequences follow:

Commodore Mode:

This is the ViNCEd operating mode which offers most compatibility to the original console handler. ViNCEd interprets all control sequences in the same way as the original handler, even if they contradict the VT-xxx standards. This is the mode ViNCEd should operate in if used as window handler for the shell, to avoid compatibility problems with existing programs. Even in this mode, ViNCEd extensions are available.

VT-220 Mode

This is the ViNCEd extended mode which interprets the control sequences in a way compatible to the VT-220 standard. It is a quite complete, but not a full VT-220 emulation. ViNCEd does not implement VT-220 sequences that conflict with the standards of the operating system, i.e. ViNCEd does not implement download-able character sets, neither user definable keymaps except thru the SetVNC program interface, nor does it come with a build-in VT-52 emulation which I regard as rather useless. However, it implements some useful extensions to the VT-220 standard found in the unix XTerm and SGI "winterm" programs, as definable window titles, selectable fonts and definable colors - all thru documented control sequences.

Scroll Region:

While ViNCEd scrolls usually the complete window contents, it is possible to separate several lines at the top and at the bottom of the window that are not scrolled. You might want to put there some global information, like the cursor position, the date, and so on, if writing an editor. The way how these lines interact with the control codes is again controlled by mode flags.

The commodore mode makes these lines "invisible" to the usual control codes. However, this is not compatible with the VT-220 definition of some control sequences and the behavour changes in this mode.

Origin Mode:

The cursor movement instructions are relative to the scrolling region in the commodore mode, i.e. the origin of the display is at the top left edge of the scrolling region. This is usually not the case in the VT-220 emulation except the Origin Mode is enabled, a standard VT-220 flag.

8 Bit Mode vs. 7 Bit Mode:

Usually, the Amiga uses eight bit wide characters. However, for some terminal purposes the eighth bit is used as a parity bit and should be ignored for that reason. To reach all ASCII-codes, use in the seven bit mode the ShiftIn and ShiftOut control codes (0x0E,0x0F) and substitute the control sequence introducer CSI 0x9B (155 decimal) by the sequence ESC [. Additionally, this substitution is also made by ViNCEd: All answer back control codes that usually come with CSI are now send with ESC [instead. Your parser code should be smart enough to handle this substitution, too.

Origin of the control sequences:

The origin of the control sequences is indicated by a character in brackets behind the description:

(C)=Commodore or standard TTY (2)=VT-220 enhancements added to ViNCEd (X)=Unix XTerm (S)=SGI winTerm (V)=ViNCEd (D)=Digital

And now for the list of the control sequences:

List of control characters

Standard TTY control characters in the range 0x00 to 0x1F, including VT-220 shortcuts in the range 0x80 to 0x9f.

List of unsupported control characters

Control characters not supported by ViNCEd.

List of ESC sequences

Control sequences introduced by ESC 0x1B, including XTerm sequences.

List of unsupported ESC sequences

ESC sequences not supported by ViNCEd.

List of CSI sequences

Control sequences introduced by CSI 0x9B or ESC [

List of CSI sequences that return results

CSI sequences that answer back with another sequence.

List of unsupported CSI sequences

CSI sequences not supported by ViNCEd.

List of sequences you might receive

The list of answer back sequences you may find in the input stream.

List of control sequences the keyboard parser sends and receives

CSI sequences that are sent by the keyboard functions .

1.443 List of control characters.

Here is the list of control codes - all codes are given in hex notation:

First the control characters in the C0 set:

07 :(C) BEL (bell), blink screen.

08 :(C) BS (backspace), move cursor backwards. Usually non-erasing, but this can be changed with ESC >?24h .

09 :(C) HT (horizontal TAB), usually only moving, but can be switched to inserting with ESC 4h .

0A :(C) LF (line feed), scroll window. Scrolling can be disabled with ESC >1h .

0B :(C) VT (vertical TAB), move upwards one line.

0C :(C) FF (form feed), clear screen and set cursor to origin. If in the commodore mode , and the cursor is not in the border, clear only the scroll region.

0D :(C) CR (carriage return), move cursor to start of line.

0E :(C) SI (shift in), set bit 7 of printed characters.

0F :(C) SO (shift out), do not set bit 7 of characters.

1B :(C) ESC (escape), the control sequence introducer.

7F :(C)(V) DEL (delete), a checkered box (a smear character), but can be turned into the forward deleting control character with ESC >?24h .

The control characters C1 in the range 0x80 to 0x9f are convenient shortcuts for more complex sequences:

84 :(C) IND (index), like Line Feed (0A), but does not place cursor at start of line.

85 :(C) NEL (next line), line feed (0A) plus CR (0D).

88 :(C) HST (horizontal tabulation set), create a TAB stop at the current cursor position.

8D :(C) RI (reverse index), move one line upwards (like Vertical TAB 0B).

9B :(C) CSI (control sequence introducer). Often used below for more complex sequences. Can be replaced by ESC [.

1.444 Unsupported control characters

The following non-printing characters are known by ViNCEd, but will be currently ignored:

8E :(2) SS2 (shift sequence 2), move to international character table G2 (Standard LATIN-1-ANSI has only the character sets G0 and G1, and ViNCEd does only support eight bit characters of G0 and G1).

8F :(2) SS3 (shift sequence 3), move to international character table G3.

9C :(2) SDC (stop downloadable character set), stops DCS sequences (ViNCEd does not support DCS sequences for downloadable character sets)

9D :(2) DSC (downloadable character set), start DSC sequence (ViNCEd does not support them)

9E :(2) Start FM sequence (ViNCEd does not support them)

1.445 List of ESC sequences

And now the list of ESC control sequences. Some of the sequences are long forms of eight bit control characters in the C1 set .

ESC is the ASCII-character 0x1B (27 decimal) and SPC is the blank space 0x20 (32 decimal); "n" is a decimal number, "strg" is a text string. BEL is the bell control code 0x07, and LF the line feed control code 0x0A (10 decimal):

ESC c :(C) global reset, clear (lower) screen. Reset colors to default, reset TAB stops, select character table G0.

ESC D :(C) like IND (84), move cursor down one line.

ESC E :(C) like NEL (85), works like LF=0A and CR=0D in one.

ESC M :(C) like RI (8D), move cursor up one line.

ESC H :(C) like SetTab (88), set tab stop at cursor position.

ESC SPC F:(2) Switch to seven bit mode.

ESC SPC G:(2) Switch to eight bit mode (default).

ESC 7 :(2) Safeback cursor, colors and style.

ESC 8 :(2) Restore colors, style and cursor position.

ESC 9 :(V) Restore colors, style, cursor and background color.

ESC Z :(2) VT-52 status request. Replied by CSI ?62;1;2;6;8;9c ("this is a VT-220 terminal")

ESC] n;strg BEL:(X) Set terminal parameters.

ESC] n;strg LF :(V) Set terminal parameters (same as above)

The value of n controls the type of the terminal parameter:

n=0 :(X) Set icon name and window title. n=1 :(X) Set icon name. n=2 :(X) Set window title. n=3 :(X) Set screen title. n=4 :(X) Like n=0 (icon & title). n=41 :(V) Set user inputs. This command inserts its string argument into the keyboard buffer as if it was typed by the user. Should be used with great care since this might easely lead to confusions. n=42 :(V) Set keymap. Hence the command echo "*E]42;d" selects the german keyboard. n=50 :(X) Change the window font. The font is given as "name.size", the appendix "font" must be dropped. As an example, echo "E*]50;topaz.8" selects the topaz font in size eight.

ESC [:(C) Replaces the CSI sequence

ESC # 8 :(D) DEC screen alignment test, fills window with E's). Used for testing.

The next ESC sequence is only understood by the CSI parser of the SetVNC buffer IO module , don't send this to ViNCEd itself:

ESC @ :(V) Insert the next control character directly.

1.446 Unsupported ESC sequences

Again, there are some ignored or unknown ESC sequences:

ESC P :(2) Introduce DCS control sequence. Since ViNCEd does not support downloadable character sets, this sequence is ignored, but accepted correctly and must be terminated with the sequence below.

ESC / :(2) Terminate DCS or APC sequence.

ESC # 3:(2) Upper half of double height and double width characters. The font is left to the system and not scalable.

ESC # 4:(2) Lower half of double height and double width characters.

ESC # 6:(2) Double width characters.

The following sequences are part of an international character support. This should be done by the locale.library together with the correct font, and not by ViNCEd itself. They are all ignored.

ESC (:(2) Select G0 character set with next character. As for ViNCEd, the G0 set is fixed to plain ASCII.

ESC) :(2) Select G1 character set with next character. As for ViNCEd, the G1 set is fixed to ECMA 94 Latin 1 graphic characters, i.e. international characters.

ESC * :(2) Select G2 character set with next character.

ESC + :(2) Select G3 character set with next character.

ESC | :(2) Select G3 set as GR (LS3R)

ESC } :(2) Select G2 set as GR (LS2R)

ESC ~ :(2) Select G1 set as GR (LS1R)

ESC n :(2) Shift to G2 characters.

ESC o :(2) Shift to G3 characters.

ESC N :(2) Single shift to G2 character set.

ESC O :(2) Single shift to G3 character set.

ESC P :(2) Introduce DCS control sequence.

ESC _ :(2) Introduce APC sequence.

ESC ˆ :(X) Introduce privacy message

The last three are all terminated by ESC /.

ESC < :(2) Enter VT-52 emulation mode. ViNCEd does not support a VT-52 emulation.

ESC = :(2) Keypad mode enabled.

ESC] 46;arg BEL:(X) Change logfile to "arg". Due to limitations set by the dos library ViNCEd can't support log files.

1.447 List of CSI sequences

The complete set of CSI sequences follow. They are all introduced by the CSI code 0x9B (=155 decimal), which can be replaced by ESC [. As above, "n" is again a number which can be dropped and defaults to one, unless otherwise stated.

The character code of ESC is 0x1B (decimal 27), SPC is again the white space, 0x20 (decimal 32).

CSI n @ :(2) Insert n spaces

CSI n A :(C) Move cursor n lines up. In the VT-220 mode , you're able to leave the scrolling region with this command, in the Commodore mode this scrolls if the cursor reaches the border of the scroll region.

CSI n B :(C) Move cursor n lines down. Same rules as above for the VT-220 mode and Commodore mode.

CSI n C :(C) Move cursor n characters forwards. Unlike formulated in the VT-220 standard, n may be zero or negative and moves in that case the cursor in the opposite direction or doesn't move it at all.

CSI n D :(C) Move cursor n characters backwards. The argument n is again allowed to be zero or negative.

CSI n E :(C) Move cursor n lines down, set cursor to the leftmost position. n can be zero or negative.

CSI n F :(2) Move cursor n lines up, set cursor to the leftmost position. Zero or negative n are allowed.

CSI n;n H :(C)(V) Move cursor to the specified row and column. n counts from one up (ugly, but true). In the Commodore mode , the cursor cannot be placed outsite of the scrolling region, and the position is relative to the scrolling region.

As an extension, each numeric argument can be preceded by >?, which moves the cursor to an absolute position ignoring the scroll region.

The roles between the modified and unmodified version of this sequence are interchanged in the VT-220 mode; the unmodified version moves the cursor to an absolute position, the modified to a position relative to the scrolling region. This default can be again overridden and reversed by the sequence CSI ?6h which selects the VT-220 "origin mode", see below.

As an extension to the VT-220 standard, negative values are accepted and specify positions relative to the bottom or right border.

CSI n;n f :(C)(V) Usually like CSI n;n H, but the details are again a bit tricky. In the Commodore mode , CSI n;n f can be used to place the cursor outside of the scrolling region, which is impossible with the former sequence. In the VT-220 mode the position is clipped to the scrolling region unless the origin mode CSI ?6h is active.

The possibility of adding >? and use of negative numbers works like with CSI n;n H above.

CSI n I :(C)(2) Move to the next nth TAB stop. The argument n is an extension and is allowed to be negative to move backwards.

CSI n J : Erase in display; n specifies the operation in detail.

n=0 :(C) (default) Clear starting at cursor position up to the end of the window.

n=1 :(2) In the CBM compatibility mode, same as n=0. In the VT-220 mode: Clear from the beginning of the window up to and inclusive the cursor position, does NOT touch the lines in the upper display region . This is a workaround for a bug in the editor "Ed". (Which guy at CBM did not read the docs?)

n=2 :(2) Clear all lower lines , do not move the cursor.

n=3 :(V) Same as n=1, but works even in CBM mode like the VT-220 control code.

In the Commodore mode , the cleared region depends on whether the cursor is in the scrolling region or in the border. In the later case, everything is cleared, in the former only the scroll region is affected.

CSI n K : Erase in line; n specifies again the operation in detail.

n=0 :(C) (default) Clear everything under and behind the cursor position up to the end of the line.

n=1 :(2) In the Commodore compatibility mode, like n=0. Again, this is a workaround for a bug in the system editor "Ed". In the VT-220 mode clear characters from the start of the line up to and inclusive the cursor position.

n=2 :(2) Clear the whole line, but do not move the cursor.

n=3 :(V) Clears characters from the start of the line to the cursor position, like n=1. But this one works even in the commodore mode.

CSI n L :(C) Insert n lines. This does usually scroll lines into the lower buffer unless the control flag CSI >?14h is set.

CSI n M :(C) Delete n lines. Scrolls in lines in the lower buffer unless CSI >?14h is set. Inserts blank lines at the bottom of the window in this case.

CSI n P :(C) Delete n characters

CSI n S :(C) Scroll up n lines. In the Commodore mode , this scrolls only the scroll region if the cursor is not in the border.

CSI n T :(C) Scroll down n lines. Again, this will only scroll the scroll region if the cursor is not in the border and ViNCEd is in the Commodore mode .

CSI n W : Cursor TAB control; details are controlled by the argument n:

n=0 :(C) (Default) Set TAB stop at cursor position n=2 :(C) Clear TAB stop at cursor position n=5 :(C) Clear all TAB stops

CSI n X :(2} Overwrite next n characters with blank spaces

CSI n Z :(2)(V) Cursor n TABs backwards, n may be negative to move forwards.

CSI n v :(V) Saveback modes. Saves back one or more of the mode control bits that can be set with CSI n h and cleared with CSI n l. For the allowed values of n, see there. (sort of DEC control sequence that conflicts with another sequence and is therefore here).

CSI n w :(V) Restore mode. Restores one of the mode control bits usually set by CSI n h or CSI n l. See there for more information. (sort of DEC that conflicts with with another sequence at the same position.)

CSI n h :(C)(2)(V)(D) Set mode control flag or flags. Like with CSI n l, CSI n v and CSI n w, more than one flag is allowed here at a time, separate the flags you want to set by a semicolon ";". See CSI n l for all available flags.

CSI n l :(C)(2)(V)(D) Clear mode control flag or flags. If more than one flag is given, the flags must be separated by a semicolon ";".

Here the list of legal (known) values. The default is printed in brackets, but for some mode flags, you may change the default with the SetVNC program as well. Just follow the links to find out how the flag is called and select there "Contents" from the line of the gadgets above to go to the online help of the SetVNC page that controls the flag.

>?30 :[l](V) XTerm/CON: cursor mode . If this mode is enabled, the cursor won't move when you mark blocks or use the scrollers at the edges of the window. Instead, the window will "pop back" to the original position whenever you type a key or something gets printed. This is the way how XTerm behaves; the cursor can still be moved explicitly.

>?29 :[l](V) Special key parsing disable. If set, some keyboard functions of ViNCEd are disabled: History, Break (Ctrl-C thru CTRL-F), Stop (CTRL-S), all TAB expansion functions. Function keys, Quit and "Send Inputs" remain intact. This is useful for writing an editor, where you might even want to send CSI >?28h as well.

>?28 :[l](V) Send disable. If set, even the "Send Inputs" function is no longer available, together with "Quit". The close-gadget, however, remains operational. Do not send this to a shell or you're lost!

>?27 :[l](V) User block control. Block operations are not executed, but send as CSI-sequences to the input stream; they must be handled by the serving program itself by calling the vnc.library. See below for special codes you may receive.

>?26 :[h](V) Raw control. The break keys Ctrl-C to Ctrl-F and the job control key Ctrl-Z, as well as "Xon" Ctrl-Q and "Xoff" Ctrl-S are disabled in RAW-mode; their ASCII code is just send to the input stream.

If this flag is set to "l", the functions are "executed" by ViNCEd, if set to "h", they are passed to the console output = the program input stream. This flag is ignored in cooked mode .

Due to a strange "feature" of the RAW: handler, ^C thru ^F are always executed AND sent, even if this flag is "h". I don't know what this is supposed to be good for, but VNC emulates this behaviour in the CBM compatibility mode. This feature can be turned off either with CSI >?23l or by enabling the VT-220 emulation.

(Replaces somehow the missing ioctl() call of unix)

>?25 :[h](V) AutoPaste . If set, ViNCEd automatically loads a block from the clipboard if the user wants to paste text into the window. If disabled, a request is send to the input stream.

However, marking and copying of text works as usual.

>?24 :[l](V) Erasing BS . If set, the control code BS=08 erases characters and does not only move the cursor. Additionally, the control code DEL=7F (127 decimal) works like the keyboard Del-key: It erases forwards and is no longer interpreted as a printable character.

>?23 :[h](V) "RAW break bug emulation". If set, the break functions ^C thru ^F are always executed, regardless of whether the CSI >?26 flag is set or not. In worst case (and that is what the flag defaults say), the break keys send an ASCII value AND generate a break signal. This workaround can be disabled either by turning on the VT-220 emulation, or by setting this flag to "l".

>?21 :[l](V) Bottom adjust . In turned on, keep the last line of the window adjusted to the bottom of the window if the window gets resized. This is the behavour of the old CON: window. If disabled, ViNCEd will use its default procedure, i.e. to insert blank lines or lines from the lower buffer.

>?19 :[h](V) Wide window . Do not wordwrap at the right border of the window, but scroll. ViNCEd does not reformat the text in case the window gets resized.

>?18 :[h](V) Follow print . Follow the cursor while printing by scrolling the window to make it visible.

>?16 :[l](V) AutoIndent. After a "Send Inputs" or a "Split" function , place the cursor under the first non-space of the line above. Useful for editors, useless for the shell; do not set it.

>?15 :[l](V) SevenBitReports. If enabled, ViNCEd will report all keyboard sequences and all answer back sequences as if the seven bit mode has been enabled. This might help some VT-xxx terminals to get the cursor keys right. To be precise, this is a workaround against a bug in some "termcap" libraries since ESC [and CSI should be considered to be identically. Unfortunately, they are not.

>?14 :[l](V) Extended bold . If this bit is set, ViNCEd uses the extended pens 8-15 for characters which would usually appear in boldface. Pen 0 is mapped to pen 8 and so on. The bold attribute controls the mapping of the foreground color, the blinking attribute that of the background color.

>?13 :[h](V) Buffer scrolling . If disabled, ViNCEd does not scroll lines into the lower display buffer in case blank lines are inserted into the window. If lines get removed, only blank lines are inserted into the window if this flag is disabled and the lines below the bottom edge of the window remain untouched.

>?12 :[l](V) Shell mode . If set, TAB-expansion and job control (Ctrl-Z) are enabled, as well as some other shell goodies .

>?10 :[l](V) Edit buffer. If set, the size of the review buffer is unlimited (well, except by memory).

>?8 :[l](V) ANSI coloring . If set, the "set render" sequence CSI m is interpreted strictly and ViNCEd will try to select colors that fit the ANSI standard as close as possible. Details how to change these colors and what can be done with this mode are in a separate section .

Changing this flag results also in a reset of the rendering colors.

>?6 :[l](V) Blinking cursor . Enables the blinking cursor.

>?4 :[l](V) Underscore cursor . Use an underscore cursor instead of the block cursor.

>?3 :[l](V) Row mode. If set, the user is not allowed to leave the current line, and ViNCEd is turned into a pure and very strict line editor. Even the "Send" function DOES NOT leave the current line, unlike what you and the shell expects. This mode is reserved for filling out "forms" and is not useful for the shell; Beware!.

>?2 :[l](V) VT-220 mode . If set, ViNCEd interprets control sequences according to the VT-220 standard and not compatible to the original console. This affects mostly the handling of the scroll region and the interpretation of the cursor placement sequences CSI H,CSI f, CSI S,CSI T, as well as some other minor interpretational differences.

?25 :[h](2)(D) Display cursor. If disabled, the cursor is invisible.

?7 :[h](C)(D) Wordwrap. If disabled, the text is not broken at the chosen border, but the additional characters are lost.

?6 :[l](2) Origin mode. If enabled, the CSI H sequence will place the cursor relative to the scroll region in the VT-220 emulation. Ignored in the CBM mode.

?5 :[h](2)(D) Reverse video . Works only with the ANSI coloring enabled as well. Reverse video will swap the ANSI black and white colors so that the default is black text on white background. The meaning of the ANSI color pens 0 and 7 is interchanged, but all other colors remain the same.

?1 :[l](2)(D) Numpad mode . Use the numeric keypad for cursor control functions.

20 :[h](C) Auto CR. If set, the line feed control code 0A (=10 decimal) includes the implicit execution of CR, i.e. moves the cursor to the left, too.

4 :[l](2) Insert mode . If set, printing inserts instead of overwrites. Does not change the handling of user input.

2 :[l](2) Keyboard lock. If enabled, the keyboard is locked and no inputs are accepted. Do not send this to the shell or you're lost!

>1 :[h](C) Scroll lock. If cleared, scrolling is forbidden.

CSI n g :(2) Tab control, details depend on n:

n=0 :(2) (default) Clear the TAB stop under the cursor. n=3 :(2) Clear all TAB stops. n=100 :(S) Reset TAB stops to default.

CSI n m :(C)(V) Rendering control. More than one option can be given, separate them by ";". The restriction to ">" arguments as for the console device does not hold for ViNCEd, ">n" style arguments may go anywhere.

n=0 :(C) Plain text, default color. n=1 :(C) Enable bold. n=2 :(C) Faint (secondary color, usually white) n=4 :(C) Enable underline. n=5 :(2) Enable blinking. This version of ViNCEd does not support blinking text, it just sets the text in bold, like XTerm. This attribute makes a difference in case the mode CSI >?14h "Extended bold" is set, and selects the extended colors for the character container box.

n=7 :(C) Reversed (exchange character/cell colors). n=8 :(C) Concealed (invisible text for passwords, etc. However, the text will be still visible in files saved with the Project Menu .)

n=20 :(V) Plain text, default color. n=21 :(S) Disable bold. n=22 :(C) Default foreground color, disable bold. n=23 :(C) Disable italic. n=24 :(C) Disable underline. n=25 :(2) Disable blinking. n=27 :(C) Disable reverse. n=28 :(C) Concealed off. n=30..37:(C) Set foreground color to 0..7, or select ANSI colors 0..7, depending whether the ANSI coloring is active. Same goes for all color selection sequences below.

n=38 :(V) Set foreground color to 8 n=39 :(C) Set foreground color to default or to the value saved with CSI SPC s

n=51 :(V) Special in the sense that this is ignored by ViNCEd itself, but used by SetVNC for buffer outputs written with the PUT argument . This sequence selects user input text.

n=59..65:(V) Set foreground color to 9..15. The current ViNCEd version can control up to 16 colors, but not more. Colors 8 to 15 are the "extended" colors.

n=40..47:(C) Set background color to 0..7 n=38 :(V) Set background color to 8 n=39 :(C) Set background color to default or to the value saved with CSI SPC s

n=71 :(V) Again a private SetVNC sequence. Selects printed text.

n=79..85:(V) Set background color to 9..15

REMARK: If you activated the ANSI rendering scheme with the CSI >?8h sequence, the arguments to CSI m to not specify a pen number (as with CON:), but a predefined ANSI color.

More rendering-options:

>n :(C) Set fill color to n, n=0..15. Unlike the clumsy console.device, this might appear anywhere in the control string.

?n :(V) All values of n presented above are valid, but only change the appearance of the user input; the style of printed characters remain intact.

REMARK: As a special hack, if no argument is given at all, the text rendering is reset to the defaults. DO NOT DEPEND ON THIS WORKING IN NEWER RELEASES, this is a compatibility hack.

CSI n;n r :(2) Set scroll region. This sets the size of the scroll region. All lines above and below this region are not scrolled, thus, kept where they are. The first argument is the first line of the scroll region, "1" being the topmost line. The second argument is the last line of the scroll region. At least two lines must be in the region or it gets disabled. If the VT-220 mode is enabled, the cursor gets set to the left top edge of the window, or to the first line of the scroll region if the "origin mode" CSI ?6h is active.

CSI SPC s :(C) Set default rendering. (SPC is the blank space, ASCII 32=0x20). Sending this sequence sets the rendering type and color defaults to the currently active pens and draw style. This default can be restored later by "CSI 0 m", "CSI 29 m" or "CSI 39 m". However, the defaults get lost on a full reset "ESC c" and are then re-read from the DrawInfo of the screen ViNCEd displays its window on. If you enabled the ANSI coloring , the default is white text on black background, as defined by the ANSI standard, unless the reverse video mode CSI ?5h is active.

CSI "62;n p :(2) Selects the bit width, details depend on n

n = 1 : 7 bit mode (8th bit set to zero) n = 0 : (default) 8 bit mode

CSI "61p :(2) 7 bit mode. Enables seven bit mode.

CSI !p :(2) Like ESC c, full reset.

CSI n p :(C) Cursor control, details depend on n

n=0 : Disable cursor : Everything else, including no argument at all: Enable cursor.

Some Amiga specific control codes follow. They have all in common that in the case of a missing argument the default behaviour is re-established.

CSI n t :(C) Set page length to n lines. The remaining part of the window can be used for drawing graphics on yourself and is not touched by ViNCEd. It is, too, not refreshed on a resize, that's up to the program drawing in the border.

CSI n u :(C) Set line length to n characters. The remaining part of the window won't be touched by ViNCEd and can be used to display graphics.

CSI n x :(C) Set left offset to n pixels.

CSI n y :(C) Set top offset to n pixels.

The next control sequences are ViNCEd specials:

CSI reg,r,g,b V:(V) ViNCEd set color sequence.

This sequence will change the screen colors of ViNCEd. They work only if the ViNCEd window was opened on its own screen, as told by the parameters in the window path .

These screen colors can be set by default as well, using the second window page of SetVNC.

Colors defined in the defaults can be reset, i.e. the CSI V can be un-done with the reset control sequence ESC c .

"reg" is the index of the color register you want to change. Currently, ViNCEd supports only 16 different color registers.

"r,g,b": The red, green and blue component of the color to load into "reg". The values are given as 16 bit integers, i.e. 65535 is the maximum.

If "reg" is between 16 and 31, the color register "reg"-16 is affected; thus, "reg"=16 and "reg"=0 set both the background color. However, the red, green and blue components are now given as four bit integers, i.e. 15 is the maximum. This gives a coarser color specification, but the numbers are somewhat more readable.

If "reg" is between 32 and 47, this sequence selects a color for the ANSI pen "reg"-32, hence overriding the ViNCEd defaults; this works even if no custom screen was opened for this window.

The color itself is given as 16 bit values "r,g,b".

To specify four bit ANSI colors, use values for "reg" between 48 and 63, again selecting the color of the ANSI pen "reg"-48.

A value between 64 and 95 releases the ANSI pen "reg"-64. This does NOT mean that some colors are reloaded with system defaults, it is just the pen that is released; a different program running on the same screen will be allowed to load and change it.

CSI ?reg,r,g,b V:(V) ViNCEd set cursor color sequence; note the question mark in front of the register number.

reg = ?1 : Use a colored cursor, use 16 bit color definitions, i.e. 65535 is lightest.

reg = ?17 : Use a colored cursor, specify colors in four bit values, i.e. 15 is highest color intensity.

reg = ?33 : Use a colored cursor, specify 16 bit color values but load the color with a higher precision, i.e. specify a higher priority for the cursor color. This is useful if you want to load the cursor color register with a definite value even though you're working on a screen with very few pens available.

reg = ?49 : Use a colored cursor, define the color as four bit value, use higher precision.

reg = ?65 : Use the default "COMPLEMENT" style cursor, free the cursor color. THIS DOES NOT MEAN that the color registers are loaded with the default screen values you found there before you colored the cursor. This does ONLY mean that ViNCEd releases the control over the color register back to the operating system and uses the old style cursor instead.

CSI n Y :(V)(S) ViNCEd window manager control. Details depend on n.

This is a replacement for a similar SGI winterm control sequence that does not obey the ANSI rules and hasn't been implemented in ViNCEd for that reason. This control sequence performs some intuition related actions:

n = 0 : Activate this window. n = 1 : Send window to front. n = 2 : Send window to background. n = 3 : Send the screen this window resides on to the front. n = 4 : Send the screen to the background. n = 5 : Show the screen title. n = 6 : Hide the screen title.

The values n=5 and n=6 are only working if the window path forced ViNCEd to open its own screen. They make only sense for BACKDROP like windows and send the screen drag bar behind or in front of a BACKDROP windows. Hence, this action can't be seen, usually.

1.448 Sequences that return results.

Some CSI sequences not only set some parameters, they also send back some kind of CSI sequence to inform the user of the result. For example, you may ask ViNCEd to send the current dimension of the window and much more...

The list of the answer back messages can be found in a separate section .

CSI n { :(C) Set Raw Events. n is a list of input events that should be send to the input stream. Unlike with CON:, this works in all console modes , even in cooked mode, but if in raw mode, all selected events will be disabled when switching back to cooked mode. This is again a compatibility hack, to make old programs working. As a second hack, the window-close input event is always active when switching to raw mode, even if you did not request it. This was done to make the program "more" working (sigh). If you do not want to receive the window-close event in raw mode, you must explicitly turn it off. A third special event is the timing event: while this event never worked in CON: windows (try it !), it works in ViNCEd. Since your input stream will be trashed with a lot of timing events otherwise, this event gets disabled as soon as you receive a timing event. You have to re-activate it again, each time. For more about the input events, consult the RKRM-Devices and check the list of answer back sequences to find out which type of input you receive.

CSI n } :(C) Reset Raw Events. The selected Raw-Events are disabled again.

For the next control sequences, "n" is a literal lowercase n, and not a number:

CSI 6 n :(C) Send cursor position, relative to scroll region if in Commodore mode, absolute in VT-220 mode.

CSI >?0 n :(V) Send cursor position, but unlike CSI 6n it is send as a string that can be send back to ViNCEd to restore the cursor position later. It is relative to the scroll region in Commodore mode, absolute in VT-220 mode if the "origin mode" is disabled.

CSI >?1 n :(V) Send absolute cursor position. Leads to CSI n;n f in Commodore mode, CSI n;n H in VT-220 mode.

CSI >?2 n :(V) Send relative cursor position (relative to scroll region), leads to CSI n;n H in Commodore mode, to CSI n;n f to VT-220 mode.

CSI >?3 n :(V) Send ViNCEd version string. Leads to CSI version;revision V

CSI >?4 n :(V) Send mode flags. Sends ESC [and a sequence of mode flags (xxx h, xxx l) that describe their current settings.

CSI 0 SPC q :(C) Send window borders, compatible to CON: (SPC is the blank space 0x20 = 32).

CSI SPC q :(C) Like CSI 0 q, but a hack to make the csh working. You SHOULD NOT DEPEND ON THIS SEQUENCE, it will be removed sometimes.

CSI >?0 q :(V) Send window borders, together with the scroll region.

CSI >?1 q :(V) Send maximal printable domain.

CSI >?2 q :(V) Send window domain.

CSI >?3 q :(V) Send window domain without scroll borders.

CSI c :(2) VT-52 status request. Send terminal ID, identically to ESC Z. Replied by CSI ?62;1;2;6;8;9c ("this is a VT-220 terminal")

CSI 5n :(2) Terminal status request. Answered by CSI " 0 n ("terminal is fine").

CSI ?15n :(2)(D) Printer status request. Answered by CSI ?11 n ("printer not ready").

CSI ?25n :(2)(D) Keyboard status report. Answered by CSI ?21 n ("User definable keys are locked")

CSI ?26n :(2)(D) Keymap status request. Answered by CSI ?27 ; 0 n ("Keyboard language unknown")

CSI >c :(2) Answered by CSI >1 ; 10 ; 0 c

1.449 Unsupported CSI sequences

Unsupported CSI sequences, partially conflicting with Commodore sequences, partially depending on 16 bit characters and other stuff I do not want to implement:

CSI n;n s :(2)(V) Set horizontal scroll region. The selected number of left and right rows does not get scrolled. This should work someday like a horizontal counterpart of CSI n;n r, is however quite tricky to implement.

CSI n T :(X) Mouse tracking control. Conflicts with the Commodore scrolling command.

CSI n x :(2) Request terminal parameters. Conflicts with Commodore's "set left offset".

CSI n q :(2) LED control. This should be done by audio programs, not by ViNCEd.

CSI n n :(S) (The second n is a literal lowercase n), with n=100 to 107. Send RGB color codes of the palette entries 0 to 7. Replaced by something smarter.

CSI n s :(2)(D) DEC save parameters. Conflicts with ViNCEd "Set Scroll Region", but functionally replaced by CSI v (see above)

CSI n r :(2)(D) DEC restore parameters. Conflicts with ViNCEd / VT-220 "Set Scroll Region", but functionally replaced by CSI w (see above)

Some standard mode flags are not supported (mode flags are arguments to CSI h and CSI l):

n=6 :(S) Lock scroll buffer

n=9 :(S) Visual bell. This is controlled by the sound preferences.

n=12 :(S) Overlay mode (2nd buffer) or duplex control. n=?2 :(2) USA characters for G0-G3 set, or VT-52 n=?3 :(2) 132 rows mode. Depends on the size of the window, not under control of ViNCEd.

n=?4 :(2) Smooth scrolling. n=?8 :(2) Auto repeat. Under control of the prefs. n=?9 :(2) Send mouse position on key press. n=?18 :(2) Send FF after printing. n=?19 :(2) Print full screen/scroll region. n=?38 :(2) Tektronix mode. Unsupported by ViNCEd. n=?40 :(2) Allow 80/132 width switching. n=?41 :(2) Curses Fix. This is Amiga, not Unix. Should be part of ixemul, not of ViNCEd.

n=?42 :(2) Enable national character set. Not under control of ViNCEd.

n=?44 :(2) Ring bell on window border. n=?45 :(2) Reverse Wraparound. n=?46 :(2) Start logging. ViNCEd does not support big brother. (-;

n=?47 :(2) Use second screen buffer. n=?1000:(X) Mouse tracking on press & release. n=?1001:(X) Hilite mouse tracking.

All tektronix sequences (not listed here) are also not supported.

1.450 Sequences you might receive

Sequences, you might receive thru your input stream additionally to the keyboard functions send to you in the english mode .

Please note that in 7 bit mode , CSI is replaced by ESC [.

CSI n;n V :(V) Version report. Consists of version and revision in this order.

CSI n;n R :(C) Cursor position. Answer of CSI 6n.

CSI n;n H :(V) Cursor position. Answer of CSI >?0n and others.

CSI n;n f :(V) Cursor position. Answer of CSI >?0n and others.

CSI n;n;n;n r :(C) Window bounds report. Answer to CSI 0q and others. The numbers are the upper left edge position and the height and width of the requested area.

CSI n SPC v :(C)(V) (SPC is the blank space 20, 32 decimal) Copy/paste report. Received if AutoPaste mode flag CSI >?25 is disabled or block control is completely under user control by the "user block control" mode flag CSI >? 27.

n=0 :(C) Paste n=1 :(V) Copy n=2 :(V) Cut n=3 :(V) Hide n=4 :(V) Select All n=5 :(V) Copy quiet (copy, but do not hide)

CSI n;n;n;n;n;n;n;n | :(C) Raw Event report. Requested by CSI {. Works also in cooked mode. Parameters are the class, subclass, the code, the qualifier, the position x and y and the seconds and microseconds of the event. For details, study the RKRMs .

The next ones can be heard only in raw mode :

CSI n ~ :(C) Function keys and other special keys.

n=0..9 :(C) Function keys F1 to F10. n=10..19:(C) Shifted function keys F1 to F10. n=20 :(2) Function key F11 (usually not available) n=21 :(2) Function key F12 (usually not available) n=30 :(2) Function key F11 with shift. n=31 :(2) Function key F12 with shift.

The next ones look unusual, but are sent by ViNCEd depending on the keymap mapping, even with a standard keyboard:

n=40 :(2) Insert n=41 :(2) Page Up n=42 :(2) Page Down n=43 :(2) Pause/Break n=44 :(2) Home n=45 :(2) End

n=50 :(2) Shifted Insert n=51 :(2) Shifted Page Up n=52 :(2) Shifted Page Down n=53 :(2) Shifted Pause/Break n=54 :(2) Shifted Home n=55 :(2) Shifted End

CSI A :(C) Cursor up CSI B :(C) Cursor down CSI C :(C) Cursor right CSI D :(C) Cursor left CSI T :(C) Shift cursor up.

CSI S :(C) Shift cursor down. CSI SPC @ :(C) Shift cursor right. (Again, SPC is the blank space) CSI SPC A :(C) Shift cursor left.

CSI Z :(C) Shift TAB

CSI ?~ :(C) Help

The next sequences can be received only in medium mode . They identify special ViNCEd features and are sent out if an external shell should run a TAB expansion and others.

CSI id;len;crs U:(V) Medium mode report. Sends this CSI sequence, and the current input line as pure ASCII of the given length "len", with the relative cursor position "crs" within this line. If "crs" is one, this identifies the cursor to be at the first character of the line, and so on. "id" specifies what to do with this line, i.e. an feature usually implemented by ViNCEd, which should be done now by the user shell:

12 : first Tab expansion 13 : first Tab expansion reverse 22 : second Tab expansion 23 : second Tab expansion reverse 32 : third Tab expansion 33 : third Tab expansion reverse 42 : fourth Tab expansion 43 : fourth Tab expansion reverse 52 : fifth Tab expansion 53 : fifth Tab expansion reverse

2 : history upwards 3 : history down 4 : search history upwards 5 : search history downwards 6 : recall history upwards 7 : recall history downwards 10 : rewind history

Except for these control codes, ViNCEd will also send the code 1A in case the Ctrl-Z function is used in the medium mode instead of executing it.

Another sequence you can hear in english and medium mode is the line-feed character 0A. It is send whenever a new prompt must be displayed and the mode flag >?29 is high.

The next sequences are answer back sequences of a VT-220 terminal which can be received by ViNCEd as well, provided the proper control codes are send to it.

CSI ?62;1;2;6;8;9c: "This is a VT-220 terminal". Answer of CSI c and ESC Z. CSI " 0n : "Terminal is fine". Answer of CSI 5n. CSI ?11n : "Printer not ready". Answer of CSI ?15n. CSI ?21n : "User definable keys are locked". Answer of CSI ?21 n. CSI ?27;0n : "Keyboard language unknown". Answer of CSI ?26 n. CSI >1;10;0c : Answer of CSI >c.

1.451 List of control sequences the keyboard parser sends and receives

This is the list of the CSI sequences the keyboard parser knows. You shouldn't print these sequences since they won't work if they are found in the output stream. These control sequences are only useful in two situations:

o) In case you've selected the english console mode , these sequences will be send to your input stream and encode the enhanced keyboard functions of ViNCEd.

o) You send a string to the keyboard parser directly. That can be done either by the ESC] 41; escape sequence , by the vnc.library function DoAsciiData() or by using the dos packet ACTION_SETLINE. This string will be parsed as if it has been typed in by the user. These functions are, for example, used by the TAB expansion.

The following definitions are used in the table below:

n a decimal number, represented as an ASCII string. CSI CSI control sequence introducer, hex 9B (decimal 155). Can be substituted by ESC [. ESC Escape character, hex 1B (decimal 27). SPC ASCII blank space hex 20 (decimal 32).

The character in brackets behind the sequence specifies the origin of this sequence:

(C) Commodore, VT-xxx (2) VT-220 standard (V) ViNCEd extension

The characters in square brackets indicate special properties of this keyboard function:

T does not abort TAB expansion S does not snap back the window position to the cursor I executed immediately in type-ahead mode H does not clear the history search buffer

...and now for the list of keyboard sequences; some of them have replacement sequences; these are indicated in the line below the definition. Numeric values, as the ASCII values of control characters, are given in hexadecimal.

CSI n D :(C) Cursor Left Even though this looks like the VT-220 output sequence, it works slightly different. It does not move the cursor across lines. "n" is the number of positions to move, allowed to be zero or negative.

CSI n C :(C) Cursor Right Ditto.

CSI n A :(C) Cursor Up 8D

CSI n B :(C) Cursor Down 84

CSI 1 E :(V)[H] History Up

CSI 2 E :(V)[H] History Down 12

CSI 1 F :(V)[H] Search Partial Upwards

CSI 2 F :(V)[H] Search Partial Downwards

CSI 3 E :(V)[H] Search History Upwards

CSI 3 F :(V)[H] Search History Downwards

CSI SPC n @ :(C) Half Screen Left The parameter "n" repeats this command "n" times.

CSI SPC n A :(C) Half Screen Right Ditto.

CSI n T :(C) Half Screen Up The parameter "n" repeats this command "n" times. CSI 41~ :(2) Half Screen Up Identically to above, but "n" is always one.

CSI n S :(C) Half Screen Down CSI 42~ :(2) Half Screen Down

CSI 6 E :(V) To Left Border 01

CSI 6F :(V) To Right Border 1A

CSI 54~ :(2) To Top of Screen

CSI 55~ :(2) To Bottom of Screen

CSI 4 E :(V) Prev Word

CSI 4 F :(V) Next Word

CSI 5 E :(V) Prev Component

CSI 5 F :(V) Next Component

CSI 44~ :(2) Home

CSI 45~ :(2) End

CSI 7 E :(V)[S] Scroll Up CSI 51~ :(2)[S]

CSI 7 F :(V)[S] Scroll Down CSI 52~ :(2)[S]

CSI 8 E :(V) Scroll Half Screen Up

CSI 8 F :(V) Scroll Half Screen Down

CSI 10 J :(V) Send Inputs 0D

CSI 11 J :(V) Split Line 85

CSI 12 J :(V) Insert ˆJ

CSI 13 J :(V) Line Feed 0A

CSI 20 J :(V) Send Complete Line

CSI n I :(V) TAB Forwards 09 n is again the number of TABs to move. Can be zero or negative.

CSI n Z :(2) TAB Backwards n may be again zero or negative.

CSI 12 W :(V)[T] Expand Path

CSI 13 W :(V)[T] Expand Backwards

CSI 22 W :(V)[T] Expand Short

CSI 23 W :(V)[T] Expand Short Bkwds

CSI 32 W :(V)[T] Expand Devices

CSI 33 W :(V)[T] Expand Devs Bkwds

CSI 42 W :(V)[T] Expand Dirs

CSI 43 W :(V)[T] Expand Dirs Bkwds

CSI 52 W :(V)[T] Expand Icons

CSI 53 W :(V)[T] Expand Icons Bkwds

CSI 62 W :(V)[T] Expand Alt

CSI 63 W :(V)[T] Expand Alt Bkwds

03 :(C)[TSI] Send ˆC

04 :(C)[TSI] Send ˆD

05 :(C)[TSI] Send ˆE

06 :(C)[TSI] Send ˆF

CSI 13 Y :(V)[TSI] Send ˆC to All

CSI 14 Y :(V)[TSI] Send ˆD to All

CSI 15 Y :(V)[TSI] Send ˆE to All

CSI 16 Y :(V)[TSI] Send ˆF to All

7F :(C) Delete Forwards

08 :(C) Delete Backwards

CSI 2 K :(V) Delete Full Line

CSI 12 K :(V) Cut Full Line

CSI 3 K :(V) Delete Inputs

CSI 13 K :(V) Cut Inputs

CSI 5 K :(V) Delete Word Fwds

CSI 15 K :(V) Cut Word Fwds

CSI 6 K :(V) Delete Word Bkwds 17

CSI 16 K :(V) Cut Word Bkwds

CSI 7 K :(V) Delete Component Fwds

CSI 17 K :(V) Cut Component Fwds

CSI 8 K :(V) Delete Component Bkwds

CSI 18 K :(V) Cut Component Bkwds

CSI 0 K :(2) Delete End of Line

CSI 10 K :(V) Cut End of Line 0B :(C) Cut End of Line

CSI 1 K :(2) Delete Start of Line

CSI 11 K :(V) Cut Start of Line 15 :(C) Cut Start of Line

CSI 9K :(V) Delete End of Display

0C :(C) Form Feed

CSI 20 K :(V) Clear Screen

CSI 2 v :(V) Cut

CSI 1 v :(V)[TS] Copy

CSI 0 v :(C) Paste

CSI 3 v :(V)[TS] Hide

CSI 4 v :(V)[TS] Select All

CSI 5 v :(V)[TS] Copy Quiet

CSI 18 W :(V) Reset

CSI 28 W :(V) Full Reset

CSI 30 W :(V)[TS] Toggle Esc

CSI 20 W :(V)[TS] Toggle Numlock

CSI 40~ :(2) Toggle Overwrite

13 :(C)[TSI] Suspend

11 :(C)[TSI] Resume

CSI 14 W :(V)[S] Abort Expansion

CSI 0 E :(V)[T] Scroll to Cursor

02 :(C) Rewind History

19 :(C) Yank

1C :(C) Generate EOF

CSI n~ :(C) Function key "n", or function key "n" shifted. Check the received CSI sequences list for details.

07 :(C) Display Beep

CSI 43~ :(V)[TSI] Toggle Pause

CSI ?~ :(C) Help

CSI 17 Y :(V) Fork New Shell

CSI 21 W :(V) Insert CSI

CSI 31 W :(V) Insert ESC

1.452 ANSI Colors

By using the "set render" CSI sequence , you're able to specify colors - or more precisely - pen values in which the text has to be rendered. However, how the arguments of the CSI m sequence are used depends on whether the "ANSI coloring" is enabled or not.

In the default mode - which is compatible to the old console device - the color numbers mean simply the hardware pen to use to render the text, whichever color has been loaded into the hardware register.

However, if you select the "ANSI coloring", these arguments specify one of the sixteen pre-definable color values and are not related to the hardware register used to display this color on the screen. One possibility to choose these colors, which is also the default, is to use the pre-defined standard "ANSI colors". Additionally, the standard color layout is chosen - to match the ANSI specifications, namely text in pen 7, white, on background in pen 0, black.

The ANSI coloring is available either as default , or as an option in the open path , or can be selected by an CSI sequence , online.

The following table gives an overview about the ANSI color definitions and how they are activated. The first eight colors are compatible to the ANSI standard; you may, however, still define them yourself, check the second window page of SetVNC. The colors 8 to 15 are ViNCEd extensions to the standard.

ANSI Color Activated by for foreground for container for background

0 black CSI 30 m CSI 40 m CSI >0 m CSI ?30 m CSI 40 m

1 red CSI 31 m CSI 41 m CSI >1 m CSI ?31 m CSI ?41 m

2 green CSI 32 m CSI 42 m CSI >2 m CSI ?32 m CSI ?42 m

3 yellow CSI 33 m CSI 43 m CSI >3 m CSI ?33 m CSI ?43 m

4 blue CSI 34 m CSI 44 m CSI >4 m CSI ?34 m CSI ?44 m

5 magenta CSI 35 m CSI 45 m CSI >5 m CSI ?35 m CSI ?45 m

6 turquoise CSI 36 m CSI 46 m CSI >6 m CSI ?36 m CSI ?46 m

7 white CSI 37 m CSI 47 m CSI >7 m CSI ?37 m CSI ?47 m

The arguments in the range 30..37 set the foreground color, the range 40..47 is responsible for the background color. The sequences of the type "CSI > _ m" select the window fill color. The similar control sequences with the additional question mark "CSI ?_ m" set the color for the user input only.

The following pens are defined for ViNCEd, and are NOT ANSI standard, nor CON: compatible. They work ONLY in ViNCEd windows:

ViNCEd Color Activated by for foreground for container for background

8 grey CSI 38 m CSI 48 m CSI >8 m CSI ?38 m CSI 48 m

9 pink CSI 59 m CSI 79 m CSI >9 m CSI ?59 m CSI ?79 m

10 spring green CSI 60 m CSI 80 m CSI >10 m CSI ?60 m CSI ?80 m

11 orange CSI 61 m CSI 81 m CSI >11 m CSI ?61 m CSI ?81 m

12 sky blue CSI 62 m CSI 82 m CSI >12 m CSI ?62 m CSI ?82 m

13 violet CSI 63 m CSI 83 m CSI >13 m CSI ?63 m CSI ?83 m

14 neon green CSI 64 m CSI 84 m CSI >14 m CSI ?64 m CSI ?84 m

15 light gray CSI 64 m CSI 85 m CSI >15 m CSI ?64 m CSI ?85 m

This ANSI coloring works ONLY for Kickstart releases 3.0 (39.xx) and up since the necessary system functions are not available in earlier releases.

REMARK: You may alter the ANSI color layout, i.e. the colors that ViNCEd chooses for the ANSI pens. This works either with the private CSI sequence "CSI V" or by using the prefs editor SetVNC , the second "Window" page.

REMEMBER: ViNCEd TRIES its best to find colors that look like the definitions above, but this isn't always possible. Don't expect miracles, on a four color screen screen you'll only see four different colors, not more! If there isn't a red color, then... THAT's IT.

The ANSI colors are mapped to the most similar looking screen pens, regardless if they look "similar" at all. ANSI "red" might get mapped into grey if nothing "more red-ish" is available.

ViNCEd uses shared pens for the ANSI coloring, but tries to allocate more pens if no matching color can be found. Remember that this might be impossible if either the screen has a very low depth or other programs running on the same screen allocated pens as well.

If you want all ANSI colors, or as much as possible, open ViNCEd on a private or public screen, read the window path section on how to do this. Then define colors to be used by ViNCEd on this screen, by using the preference editor, again on the 2nd window page .

ViNCEd might choose not to allocate additional pens for the ANSI colors if "sufficient" matches are found. This depends not only on if more pens are available, but also on the depth of the screen. ViNCEd is more tolerant on a screen of low depth! For example, ViNCEd usually won't allocate ANY ANSI colors on screens of depth three (eight pens) or below, leaving the additional pens to other purposes. IF you want to use ALL pens as ANSI pens, YOU HAVE TO DEFINE THE COLORS EXPLICITLY, again with SetVNC .

I would suggest to leave the first four colors alone, and fill in the remaining colors to get a perfect ANSI match. If, for example, you're working with the standard CBM workbench layout, a blue, white, grey and black color is already available. Define colors four and up as the missing colors, namely red, yellow, green, magenta and turquois for a full ANSI set, and additionally light blue, pink, neon green, spring green, orange, dark green and violet for a full ViNCEd set of colors.

1.453 Console Modes

For short, ViNCEd has four where the standard terminal has only two.

But I guess I should first give an explanation what is this about:

The standard Amiga Os comes with two "flavours" of window handlers, the "CON:" handler and the "RAW:" handler. Both are actually handled by the same program and you can therefore convert a "CON:" window into a "RAW:" window and vice versa. The difference between the two is how they handle input.

In "RAW" mode, each key press is send immediately to the input stream of a waiting program, and none of these key presses is echoed on the screen. It's the matter of the program of do that. "RAW" mode has very little to offer, all has to be done by the external program using the console - read key-press by key-press and decide what to do about it. This mode is, however, very adapted for running an editor in the console since it gives full control over what is printed on the screen at which time and where.

The "CON" mode is different from that. Usually, no inputs are send to the input stream of a program at all. Instead, the console lets you edit a line as a whole, including all the nice editor features of ViNCEd like delete characters, insert some, go back in the history and much more. Then, if you press Return, the collected inputs on one line are send back to the program as a complete line. The shell uses this mode since it is more convenient to leave all the troublesome editor features for the console and care only for parsing complete shell lines. This mode is also called the "canonical mode", or the "cooked mode" - because it is "not raw". The ViNCEd convention for this mode is "well done" for a reason you'll be able to understand in a second.

ViNCEd has more to offer than just "raw" and "cooked". There's something in between.

The raw mode:

This works as explained above. All key presses are directly send to the input stream, without any buffering. However, to be compatible to the standard console, no extended ViNCEd key codes are send back, only standard functions will be available. Your ViNCEd keyboard definitions won't be used by this mode. All this, as I said, for compatibility since a program might not expect some unusual CSI sequences.

The english mode:

This is "almost raw"; it works like the raw mode above, except that all the ViNCEd extended keyboard CSI sequences are send as well. An extended editor may profit quite a lot from that, as it will be able to receive sequences as "move to the home position", "toggle overwrite mode", "move to the next word" and so on. An editor using these functions need not to offer a separate keyboard editor because this is already setup and controlled by ViNCEd.

The medium mode:

This is "almost cooked"; it works like the "cooked" = "well done" mode below, except that ViNCEd does not provide the TAB expansion , history and job control ; I don't like the way how these functions are currently implemented because these are features the shell - and not the shell editor - should provide. In case any of these functions is used, ViNCEd will not perform them, but will send on a CSI sequence to tell the shell to do so; this sequence will include the complete line you've entered so far. It's the matter of the shell to extract the right information. Except for that, this mode works like the "well done" mode below.

A future "VinShell" ("Vinchy") might make use of this mode. It offers the flexibility of the "raw mode" and the convenience of the "well done" mode.

The well done mode:

This is what was usually called the "cooked mode". (However, you don't "cook" steaks, they are "well done" in case you haven't noticed the origin of the mode names ;-). All user inputs are buffered, the console provides additional functions like the TAB expansion and Job Control . The way how this is implemented must be called "a hack", but it is the only way to make this compatible to the standard Amiga Shell. It is fully backwards compatible to the old console "cooked" mode.

For experts: The console modes can be selected with the dos packet "ACTION_SCREEN_MODE". The following mode IDs have been defined in "vnc/packets.h":

VNC_RAW_MODE

Equals one. The standard "raw" mode. The same value goes for the standard CON: handler as well.

VNC_WELL_DONE_MODE

Is zero. This is the "cooked" or "well done" mode; identical to the identifier for the old CON:-handler.

VNC_ENGLISH_MODE

This is three and identifies the english mode.

VNC_MEDIUM_MODE

This is two and identifies the medium mode.

1.454 List of understood DOS packets

This is the list of packets understood by ViNCEd together with their parameters. More can be found in "vnc/packets.h"

ACTION_WRITE:

dp_Type: 0x57 dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: char * to buffer dp_Arg3: number of characters to print

dp_Res1: number of characters written, or -1 on error dp_Res2: error code

Write characters to the ViNCEd console.

ACTION_READ:

dp_Type: 0x52 dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: char * to buffer dp_Arg3: size of buffer

dp_Res1: number of characters read, could be less than Arg3, is zero if "End of File" is detected, is -1 if an error was detected dp_Res2: error code

Read characters or complete lines from the ViNCEd console.

ACTION_WAIT_CHAR:

dp_Type: 0x14 dp_Arg1: timeout period in microseconds

dp_Res1: success code; DOSFALSE if no characters present, or DOSTRUE if characters available dp_Res2: error code if dp_Res1 DOSFALSE number of inputs lines queued if dp_Res2 DOSTRUE This is an ugly hack to make the ARexx Lines() function working. Do not depend on this.

Wait a specified timeout period for characters to be available in the ViNCEd output buffer. Return DOSFALSE if the output buffer (hence your input buffer) remains empty. Returns DOSTRUE if characters are available. The number of available lines in the output buffer will be returned in dp_Res2. However, this is a hack to keep ARexx happy you should not depend on in future versions. The output buffer of ViNCEd is character oriented, not line oriented.

ACTION_DISK_INFO:

dp_Type: 0x19 dp_Arg1: BPTR struct InfoData. This structure is filled as follows:

id_DiskType either CON\0 or RAW\0 dependent on the console mode. id_VolumeNode the pointer to the intuition window used by ViNCEd id_InUse the pointer to an struct IORequest opened for the console.device

all other fields are zero.

dp_Res1: success code, DOSTRUE or DOSFALSE. You NEED to care about DOSFALSE! ViNCEd might be unable to open a window if the system is too low on memory. dp_Res2: an error code.

BIG BIG WARNING:

THIS PACKET IS A MESS. DO NOT USE THIS PACKET UNLESS YOU CALL ACTION_UNDISK_INFO, too.

This packet will pop-open the window in case it has been closed or iconified because certain programs depend on an correctly setup pointer in the structure above. Do not use the IO request structure. Sending out read or write requests in ViNCEd won't do good. ViNCEd windows are not handled by the console device. The id_DiskType is set to CON\0 (four characters packed into one long word) for the medium mode or the well done mode . It is set to RAW\0 otherwise.

This packet is a big mess. If you MUST use it, call ACTION_UNDISK_INFO afterwards. If you want to set the keyboard or the font, DO NOT MESS with this packet. Use the documented ESC codes to set these properties.

ACTION_UNDISK_INFO

dp_Type: 0x201

dp_Res1: DOSTRUE or DOSFALSE dp_Res2: error code

This packet un-does the side effects of ACTION_DISK_INFO. The window will be allowed to closed or iconified after this packet has been received. Each ACTION_DISK_INFO must be matched by one and exactly one ACTION_UNDISK_INFO.

ACTION_TIMER

dp_Type: 0x1e

Internal use only, do not send this packet type. This is intentionally undocumented.

ACTION_SCREEN_MODE

dp_Type: 0x3e2 dp_Arg1: the screen mode to select

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code in case of failure

This packet selects a different console mode for the stream associated to this packet. The following modes are available:

VNC_RAW_MODE =1 raw mode, unbuffered, single character mode, only CBM control sequences are send VNC_ENGLISH_MODE =3 english mode, unbuffered, single character mode, extended ViNCEd settings are send as well VNC_MEDIUM_MODE =2 medium mode, line buffered. TAB expansion, history, job control and other functions are no longer executed by ViNCEd, but left to the shell by sending out CSI sequences . VNC_WELL_DONE_MODE=0 the well done mode, usually known as the "cooked mode"

ACTION_FINDUPDATE ACTION_FINDINPUT ACTION_FINDOUTPUT

dp_Type: 0x3ec,0x3ed,0x3ee dp_Arg1: BPTR to struct FileHandle dp_Arg2: set this to NULL dp_Arg3: BPTR to a BCPL string containing the window path .

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

This opens a ViNCEd window. Unlike for the original CON: handler, the window path *must* be given here; the window won't be opened just because you called DeviceProc() - that's different compared to the original handler. The type specified for the stream won't matter, all three codes work alike. The following special names are available:

"*" Opens a stream to the owner of the same owner that send this packet. "CONSOLE:" Opens a stream to the NULL owner. "CONSOLE:name" Opens a stream to a named owner.

None of the two last names sends the given stream to foreground mode, so beware! You need to send ACTION_SET_OWNER to do that.

ACTION_SEEK (V)

dp_Type: 0x3f0 dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: offset dp_Arg3: seek mode

dp_Res1: absolute file position before seek operation took place or -1 on error dp_Res2: error code

No, that's not a typo. ViNCEd allows seeking in its output stream. If you seek towards the end, the data that was "sought over" will not be read and kind of "put back" for later usage. I don't know if this is useful or not, but using this Seek() operation, it is possible to determinate the number of characters in the ViNCEd input buffer. At least, this packet is implemented since VNC 1.00.

ACTION_CHANGE_SIGNAL

dp_Type: 0x3e3 dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: struct MessagePort * dp_Arg3: set this to NULL

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: struct MsgPort * to port used before if dp_Res1 is DOSTRUE, or error code if DOSFALSE

dp_Arg2 may be set to NULL to read the current port.

This command selects a port - and therefore a task - to send Ctrl-C and the other break signals to. This port must be an ACTION_SIGNAL type port.

This command is "a nice idea" but not much more; it cannot be implemented strictly. ViNCEd does also have to send the break signal to the last program that printed on the ViNCEd screen. This is an ugly hack also implemented in the original CON: handler for pre-2.0 programs. To avoid possible mess, ViNCEd checks the port to break to on every "Break" signal carefully if it is valid, but still: In case you selected a break-port with this packet and the port is about to disappear, reset the break-port by this packet to some useful value.

ACTION_NIL

dp_Type: 0x0

This is an internal packet and intentionally undocumented.

ACTION_SETLINE (also known as ACTION_FORCE) dp_Type: 0x7d1 dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: char * to a buffer dp_Arg3: number of characters

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

This packet sends characters to the ViNCEd internal keyboard buffer as if they have been typed. All ViNCEd keyboard CSI sequences may be specified here. The LF "line feed" character will be replaced by "CR", the Return key.

This packet IS multithreaded, but there is only one thread per ViNCEd window.

If you want one thread per owner, use the "ESC] 41" ESC sequence which is the recommended method.

ACTION_PUSHLINE (also known as ACTION_STACK)

dp_Type: 0x7d2 dp_Arg1: filehandle -> fh_Type dp_Arg2: char * to a buffer dp_Arg3: number of characters

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Pushes the buffer data at the end of the ViNCEd output (hence your input) buffer, in LIFO order. This is used for the ARexx "PUSH" command. However, since the ARexx DoPkt routine is buggy, this works only reliable if the ARexx fix in the ViNCEd archive has been applied. This bug is NOT ViNCEd related, it happens, too, with the original CON: handler.

ACTION_QUEUELINE (also known as ACTION_QUEUE)

dp_Type: 0x7d3 dp_Arg1: filehandle -> fh_Type dp_Arg2: char * to a buffer dp_Arg3: number of characters

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Queues the buffer data at the beginning of the ViNCEd output (hence your input) buffer, in FIFO order. This is used for the Arexx "QUEUE" command. However, since the ARexx DoPkt routine is buggy, this works only reliable if the ARexx fix in the ViNCEd archive has been applied. This bug is NOT ViNCEd related, it happens, too, with the original CON: handler.

ACTION_EXPANDLINE

dp_Type: 0x7db dp_Arg1: filehandle -> fh_Arg1 dp_Arg2: char * to a buffer dp_Arg3: number of characters

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

This packet sends characters to the ViNCEd internal keyboard buffer as if they have been typed. All ViNCEd keyboard CSI sequences may be specified here. The LF "line feed" character will be replaced by "CR", the Return key.

This is identically to ACTION_SETLINE except that the TAB expansion is not aborted, even though some "typing is done on the keyboard".

This packet IS multithreaded, but there is only one thread per ViNCEd window.

If you want one thread per owner, use the "ESC] 41" ESC sequence .

This function should be used exclusively by an alternative external TAB expansion patched into the vnc.library.

ACTION_CLEARLINE

dp_Type: 0x7dc dp_Arg1: filehandle -> fh_Type

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Removes all input from the user input line and rewinds the history.

This function uses the same thread as ACTION_SETLINE and ACTION_EXPANDLINE, so handle with care! This aborts, too, the TAB expansion.

Note: There is little use of this packet since quite the same can be done by inserting ASCII 02, the rewind history keyboard function, in the keyboard buffer with ACTION_SETLINE. Read the keyboard CSI list for details.

ACTION_SENDID

dp_Type: 0x1f8

dp_Res1: A pointer to the cn_Window structure, the main structure of ViNCEd. See "vnc/window.h". dp_Res2: The size of this structure.

Request a pointer to the ViNCEd main structure handling the current window; this structure is documented in "vnc/window.h".

Do not modify or read fields from this structure unless you've called the vnc.lib LockWindow() function to arbitrate access. This is the structure that must be passed in register a5 for most vnc.library functions. C authors might prefer to use the link library SetCNWindow() function.

ACTION_CURRENT_VOLUME

dp_Type: 0x7 dp_Arg1: filehandle -> fh_Arg1

dp_Res1: A BPTR to the struct DeviceList, the handler entry of ViNCEd. dp_Res2: set to NULL

Sending this packet does not make too much sense anyways. This is actually a file system packet that is only implemented for completeness.

ACTION_DIE

dp_Type: 0x5

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Shutdown ViNCEd completely, if possible.

This packet tries to cancel this running handler task. However, this is only successful if no streams are open, plus some other peculiarities. It returns DOSTRUE if "it thinks" the handler could be shutdown. This is not guaranteed, even in that case. If another packet is still in the input queue at the time this packet was sent, the shutdown request will be canceled, as well as by some other external occurrences.

ACTION_FLUSH

dp_Type: 0x1b

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Refreshes the ViNCEd window. Useful if scrolling is delayed and something important must be shown immediately.

This packet returns always DOSTRUE.

ACTION_ABORT

dp_Type: 0x200

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

Aborts all pending packets of the owner this packet was sent from. This makes only sense if the packets were send asynchronously, see below.

ACTION_SET_OWNER

dp_Type: 0x40c dp_Arg1: mode, see below dp_Arg2: char * name

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

This packet is actually a filing system packet; it is here abused to set the foreground ViNCEd owner .

The "mode" parameter must be zero, the "name" argument points to a C-style string (no BSTR, no BPTR!) of the name of the owner to be set to foreground. This can be NULL or a pointer to an empty string to select the NULL owner.

This packet is used by the vnc.library calls "Foreground()" and "Background()" for the job control functions.

ACTION_LIST_TO_LOCK

dp_Type: 0x202

This packet is internal use only. Do not send it.

ACTION_EXAMINE_OBJECT

dp_Type: 0x17

This packet is internal use only; it is intentionally undocumented.

ACTION_EXAMINE_NEXT

dp_Type: 0x18

This packet is internal use only; intentionally undocumented.

ACTION_FREE_LOCK

dp_Type: 0xf

Internal use only.

ACTION_COPY_DIR

dp_Type: 0x13

Internal use only.

ACTION_LOCATE_OBJECT

dp_Type: 0x8

Internal use only.

ACTION_IS_FILESYSTEM

dp_Type: 0x403

dp_Res1: DOSFALSE dp_Res2: error code, set to zero.

Check whether ViNCEd is a filing system. Answer is "no".

ACTION_SET_COMMENT

dp_Type: 0x1c dp_Arg1: set to NULL dp_Arg2: set to NULL dp_Arg3: set to NULL dp_Arg4: BPTR to BSTR

dp_Res1: success code, DOSTRUE or DOSFALSE dp_Res2: error code

This sets the window title to the supplied string; this string may contain all control characters found in the window title .

NOTE: This packet is left over from the "ConMan" handler. You should better use the documented Esc sequence "ESC] 2;" to set the window title.

ACTION_TABHOOK_RETURN

dp_Type: 0x802

Internal use only. Do not send.

ACTION_GET_DISK_FSSM

dp_Type: 0x1069

dp_Res1: NULL dp_Res2: ERROR_OBJECT_WRONG_TYPE

This packet is used for filing systems and should return the file system startup message. Since ViNCEd is no filing system, this packet fails.

Other goodies:

ViNCEd supports so called "asynchronous packets". These are packets not delivered by the DoPkt() function, but build by the vnc.library, put into the input queue of ViNCEd, that return immediately and that you can forget about. The action they perform is delayed and they get executed whenever ViNCEd has some time. These packets are sent by the vnc.library function "SendAsyncPacket()". Check the autodocs for details; do not attempt to roll these packets on your own.

Just for completeness:

These packets are identified by setting the bit 31 of dp_Type to 1.

Owner identification:

ViNCEd identifies the owner a packet was sent from by the dp_Port reply port in the DosPacket structure. ViNCEd does not provide a single port packets are collected from, but one port per owner. That means especially that you do not get the address of the handler task if you subtract 0x5c from the address of the port, as mentioned in Ralph Babel's "Guru Book". This is definitely one of his lesser smart ideas...

The method which should be preferred is to check whether mp_Flags is PA_SIGNAL, then read the "mp_SigTask" field of the port, which will always point to the handler managing the port.

1.455 What is a ViNCEd owner, please?

What are owners?

Owners are groups of processes sharing the meaning of foreground and background. A set of processes, called one owner, can be set to background, i.e. set to receive no user input and can't print to the console, while a second is put to foreground to hear the users commands. At least one owner is always present: This is the NULL owner, setup once you open a ViNCEd stream.

Owners not only share the foreground/background property, but also one "thread" of the multi-threaded ESC/CSI sequence parser. ViNCEd cannot be confused by garbled ESC-sequences that come from more than one process. (Yeah, like a real file system this part of ViNCEd is multi-threated!)

How are "owners" identified?

This is done by another new feature, "named CONSOLE"s. If a program opens a new handle to the current console window, this is usually done by the file name "*", or "CONSOLE:" under 2.0 and up. The name "*" will open a stream to the owner the opening process is already part of. "CONSOLE:", however, will identify the NULL owner.

To create a new owner that can be distinguished from the default one, use the named consoles. Open a path like

CONSOLE:name

where "name" is some unique name used to identify your process or process group. If not already created, a new owner will be constructed by ViNCEd, and your process will be added to this owner. As a default, this new owner is not put into foreground, to do this use the vnc.library functions "Foreground()" and "Background()". They receive as arguments the stream resulting in the open call.

Which owner do I get when I open a unnamed console?

This depends on:

1) whether you open your stream with "*" or with "CONSOLE:"

2) The setting of your processes "pr_ConsoleTask"s field.

If you choose the name "CONSOLE:" you get always the NULL owner that is created every time a new ViNCEd window opens - and that is usually under the control of the shell.

If you choose the "*" name, the owner belonging to the controlling console of your task is used. If opening a new owner, put the contents of the file handles "fh_Type" field into "pr_ConsoleTask", and every stream opened with "*" will belong to the new owner.

The following code segment opens a new file, creates a new owner, puts it into foreground and sets the controlling terminal to the new owner:

BPTR stream; struct Process *myproc; struct MsgPort *owner;

stream=Open("CONSOLE:myowner",MODE_OLDFILE); if (stream) { myproc=(struct Process *)(FindTask(NULL)); owner=((struct FileHandle *)(BADDR(stream)))->fh_Type; myproc->pr_ConsoleTask=owner; Foreground(stream); }

Just for the case you missed that: Creating a new owner does not bring this owner in foreground, and printing to that owner without further preparation will suspend, i.e. halt you process. You have to bring it to foreground FIRST. As a side effect, all other owners are then in background, and hence even the shell can't trash your outputs.

Another remark: The message pointer called "owner" in the code above is not really the ViNCEd-Owner structure, but if you send your packets to that port, they get identified with the right ViNCEd owner.

What actually happens when I press Ctrl-Z?

A new named owner with a unique name is created by opening a named console like above, it is send to foreground, and a new shell process is forked with the terminal process set to the new stream.

A second remark: This job control is still in a somewhat "experimental phase". ViNCEd tries to do this as safe as possible, and for that reason it is not possible to interrupt a program with Ctrl-Z if it is in an "unstable phase" of working. The job control never caused a crash on my system, but it is still a bit picky about the "stability" of the interrupted process. If you have better ideas how to do this stuff, have a look at the vnc.library, especially the entry point "CtrlZSuspend()". Write better code and patch the library, if you can!

1.456 List of Gurus thrown by ViNCEd

As every shared library, ViNCEd can produce as a last help, software failures (so called "Guru meditations"). They all start with the ID

0x1e000000

and above. This might be a bit unusual to a library if you read how these failure codes should be constructed, but it I considered converting ViNCEd into a device as a replacement for the console.device.

Here the list:

0x1e000005 ViNCEd could not find a owner that has to be removed in its hash list. 0x1e01000e The message queue lacks memory. 0x1e010001 No memory for IO structures. 0x1e01000f ViNCEd failed to create the VNCClip.supervisor. 0x1e018005 Somebody tried to free memory not under control of the ViNCEd memory manager. 0x1e038002 ViNCEd could not open the graphics.library. 0x1e038003 ViNCEd could not open the layers.library. 0x1e038004 ViNCEd could not open the intuition.library. 0x1e038007 ViNCEd could not open the dos.library. 0x1e138007 The PostRemove() procedure could not open dos.library. 0x1e03801e The handler part of ViNCEd failed to open the vnc.library. 0x1e048035 ViNCEd failed to open the input.device. 0x1e048011 ViNCEd failed to open the console.device. 0x1e060001 ViNCEd failed to install its input handlers. 0x1e060002 ViNCEd failed to remove its input handlers. 0x1e060003 Qpkt-failure: ViNCEd send a request to the timer device but received a different one. 0x1e060004 The VNCFiler.supervisor received a command from a window it does not control. 0x1e060006 The VNCFiler.supervisor received an illegal command. 0x1e060016 The VNCFiler.supervisor received an invalid message. 0x1e060013 Qpkt-failure: FindCNWindow received not its own message. 0x1e060023 Qpkt-failure: DoAsciiData received not its own message. 0x1e060026 A required TABHook packet was not available. 0x1e060033 ViNCEd received an invalid ACTION_TabHookReturn packet. 0x1e070000 The VNCClip.supervisor failed to allocate a signal bit. 0x1e070017 The VNCSaver received an illegal command. 0x1e071000 The startup code of the VNCFiler failed. 0x1e071003 The startup message to the VNCFiler.supervisor was illegal. 0x1e080001 ViNCEd could not change the window proportions to match the fonts requirements. Re-selection of topaz.8 failed. 0x1e080002 ViNCEd found an alien gadget in the windows gadget list and removed it accidentally. 0x1e08000f ViNCEd found an illegal free mem of a dynamical node.

1.457 ViNCEd goodies

ViNCEd is written as a shared library, the vnc.library, which is open to all users. A ".fd" file and the necessary link libraries are included. This library contains functions to set and read the preferences of ViNCEd, as well as controlling various ViNCEd internas as the job control . Other functions are "patchable hooks" - future enhancements might patch in here, as for example a custom TAB expansion or a custom keyboard parser.

It is quite tricky to write a handler that does not reside in L:, but is a shared library, and as such, is expungable. The vnc.library automagically mounts itself as NEWCON:, VNR: or VNC: if it gets loaded, and it automatically removes the handler from the dos device list if the library gets expunged. Thus, ViNCEd is one of the rare handlers that are expunged automatically if the system memory gets low.

Together with each window comes not only a handler process named by the device name; but, if needed, a process that controls the tab expansion and the job control, together with all other actions that are needed to interact with the DOS. A usual handler itself can't perform DOS-I/O. A second supervisor process is started by the library; the VNCClip.supervisor handles the clipboard and is a smarter replacement for the usual ConClip, which is no longer needed if work with ViNCEd. As a second job, it is also responsible for re-loading the preferences and installing them into the window.

Since ViNCEd does all the window handling by itself, all windows are by default simple refresh windows. This slows down the scrolling if a ViNCEd window is partly obscured. If you are a bit picky about this, you might want to choose the SMART window path option.

ViNCEd knows a lot more DOS packets than the old CON: handler; you may SEEK and FLUSH inside them, and the (unofficial) ARexx packets QUEUE and PUSH are supported as well. Most "ConMan" extensions are available as well, except for PIPE support which should really be implemented thru the PIPE: handler instead.

ViNCEd supports named consoles , which are a part of the unique job control mechanism. All so called "owners" are mostly independent of each other, may receive different input events , even in "cooked mode". Timer events work as well, unlike with the CON: handler.

ViNCEd windows can be opened on their own screens by giving special arguments to the window path , making tools like "ScreenShell" useless. The colors, font and the monitor ID of this screen can be setup by the prefs editor .

ViNCEd supports special control characters in the window and screen title, to display the result codes and some more information about shells running in ViNCEd windows. Check the title string section for detailed information.

The iconification gadget of ViNCEd is now replaced by an object of a public gadget class called "tbiclass" if a public class of this name is available. This should make it easier for GUI patch programs like SysIHack or MCP to replace this gadget by a custom gadget. "VisualPrefs" is the first program that supports this boopsi class and customizes ViNCEds gadgets. For details, check the "includes" drawer of this distribution. ViNCEd does now, too, support the external "toolbutton.image" class and loads it on startup.

I regard this as the "cleanest" solution that works without patching any system function at all, so please make use of it (and dump KingKong...)

ViNCEd uses smart internal memory management functions to avoid fragmentation of your system memory as much as possible. The memory manager of ViNCEd is "dynamic" in the sense that it does not only use memory pools, but also relocates memory blocks within these pools if required; therefore, memory blocks will be "moved" thru the memory to reduce fragmentation. Up to my knowledge, something like this hasn't been tried before. As you see, this works stable since Version 1.06 of ViNCEd.

1.458 Goodies of the SetVNC program

The complete configuration and support of ViNCEd is done thru one single program, SetVNC . There's no need to mess your C: directory with tons of tiny files, one is enough.

SetVNC contains, too, the user interface for the preferences editor; this interface doesn't use any big external GUI library as MUI or other horrors (a 200K GUI for a 64K program?). It's fully font sensitive; it doesn't even require "gadtools" which is too unflexible for what is needed. The GUI supports online help, you may always ask for instructions if you got stuck.

Since most of the SetVNC functions aren't needed all the time, the program is "overlayed", i.e. only the part needed is loaded into memory. This helps conserving memory and avoids disk trashing, especially on "low end" machines.

SetVNC offers options to save the screen and history buffers of ViNCEd in a fully system-friendly way, without the requirement for any hacks.

1.459 Frequently asked questions

This section contains frequently asked questions about ViNCEd. In case of a problem, please study this chapter first; I'm easely annoyed by answering these questions over and over again. Expect an "angry reply" in case you ask any of these again...

ViNCEd seems sometimes to "hang" if I press Return.

ViNCEd doesn't hang, for sure. What most likely happened is that you've typed rather fast and haven't released the "Shift" key yet when pressing Return. The Shift Return keyboard combination is, however, bound to the "Line Feed" keyboard function . This function inserts just a line feed but nothing else, it does not send any data to the shell for execution.

Solution:

To avoid this, you could simply remove this keyboard binding with SetVNC, using the keyboard page of the preferences editor.

How do I avoid moving the cursor by the cursor keys?

ViNCEd is a full screen editor, which means you're free to move the cursor anywhere you want. Why do you expect that a shell works different that your editor which uses the same keyboard layout? I'd consider this as an improvement rather than a drawback, but whatever:

Solution:

Reconfigure the keyboard, using the keyboard page of the preferences editor program, SetVNC . Bind the cursor keys "Up" and "Down" to the keyboard functions "History Up" and "History Down" and save your changes.

How to run the TAB expansion with the TAB keys?

The default preferences bind the TAB expansion functions to Ctrl TAB, mostly for backwards compatibility to CON: which does not know any form of TAB expansion . Furthermore, to enable the TAB expansion, the window must be in Shell Mode or any shell specific extensions won't be available. This mode can be selected either by the system preferences, on the first shell page , or by the SHELL window path argument.

Solution:

Reconfigure the keyboard, using the keyboard page of the preferences editor program, SetVNC . Bind the TAB key to the keyboard functions "Expand Path". More than one TAB expansion is available, so you might bind other combinations to the various TAB expansion functions as well.

How do I avoid that the cursor changes its position when I scroll the window?

One of the ViNCEd concepts is to keep the cursor always visible on the screen since you should know where you're typing. But whatever:

Solution:

Turn on the "XTerm/CON: cursor mode" flag on the first editor page of the preferences editor SetVNC , then save your changes.

I don't want to be able to set the cursor position with the mouse keys.

Some folks ask really strange questions, do they? (-: They DO NOT want to be able to do something.... But anyways:

Solution:

Turn on the "XTerm/CON: cursor mode" and the "Rigid XTerm cursor" flags on the first editor page of the preferences editor SetVNC , then save your changes.

ViNCEd messes up my output, as for example the dump of the "lha" program and others.

Solution:

Some of the preferences flags ARE NOT compatible to shell usage, these are marked as such in this guide. Do not play with settings you do not understand, check this guide first. Especially, amongst the flags you shouldn't set are the following:

"VT-220 compatibility mode" "Destructive DEL and BS" "Insertion mode for DOS output" "Notify DOS about Paste"

All of them are found on the second system page , do not play with these.

The TAB expansion does not work!

Solution:

Grumpf! Yes, it does! It might work a bit different to what you expect, however. But this is more or less a matter of the correct configuration. Did you try the TAB expansion tutorial first?

SetVNC doesn't save the preferences.

Solution:

Yes, it does. However, it does not change the preferences of the windows already open. It is not possible nor desirable to change the settings of windows already open since these settings are now under the control of the programs running in these windows. However, the new settings will be used for all windows opened after the new settings have been saved.

How to setup ViNCEd as a VT-220 terminal?

Solution:

The VT-220 flag in the preferences editor alone is not enough. It's usually required to set some more flags. Here's my selection:

CSI >?26h to send the ˆC to ˆF keys rather than to execute them.

CSI >?19l to break the lines at the right window edge.

CSI >?15h to enable seven bit answer back messages required by most termcap settings.

CSI >?13l to disable scrolling into the display buffer.

CSI >?2h to enable the VT-220 interpretation of the CSI sequences as well as the emulation of some VT-xxx illnesses.

That's about it. To enable all at once, the following command could be used:

echo "*E[>?26;>?15;>?2h*E[>?19;>?13l"

For details what all this does, check the list of CSI sequences .

I don't like the iconification gadget image.

Solution:

There's a documented way to replace it. For example, try the "VisualPrefs" program which will allow you to reconfigure it. The ViNCEd style is adapted from the unix "Mwm" window manager style I work all the day with, that's why. This archive comes, too, with the tool button image, the standalone class for the iconification and related gadgets in the window title. Probably, try to put the file Extras/toolbutton.image to SYS:Classes/Images.

When using ViNCEd as a terminal driver, Control C is never transmitted to the terminal at the other side.

Solution:

The problem is what to do with a ˆC signal at all. The Amiga standard is to break programs, even if that program has turned the console into the "RAW" mode, and not to deliver a ˆC at all. The same problem appears for Unix, but there's a system call available that controls the delivery of these special keys, namely ioctl(). The Amiga Os doesn't support this technique, but ViNCEd does: The following control sequence will turn off the signalling and will deliver all control characters literally:

CSI >?26h (see the list of CSI sequences)

The following command in the shell will do that for you:

echo "*E[>?26h"

How to turn off the horizontal scroller in the window?

Solution:

Check the third window page of the SetVNC preferences program.

How to avoid that ViNCEd prints text into the right window border?

Solution:

You can tell ViNCEd to break lines at the right window edge, check the first system page of the preferences editor. ViNCEd will NOT, however, reformat the output in case the window gets resized. This has certain reasons:

First, ViNCEd operates line-oriented, not paragraph-oriented. Breaking a line at the right border breaks it there, and there are then two lines, not one.

Second, this reformatting algorithm would be tremendously complicated. There is quite a difference between a pure line editor as CON: and a full screen editor as ViNCEd. I have to consider quite a lot of more cases than CON: which can just re-print its buffer in this case. The same does not hold for ViNCEd.

There are certain options and CSI sequences that define a scrolling region and address absolute character positions on the screen. These CSI sequences would be unuseful and their settings would become invalid in case the window gets resized. After all, I doubt there is a way of reformatting the window input AND staying compatible to the VT-220 standard. I've thought quite a lot about this problem, but the current answer is "No". The current ViNCEd version behaves quite a lot like the unix XTerm, and for good reasons.

I can't invoke the online guide!

Solution:

You didn't use the installer script that setup all variables correctly, or changed your configuration or erased the icon of the guide. You should check whether:

- The "Get Help" system macro on the third system page is setup correctly. It should say:

"SetVNC Help\r"

- The position of the online guide is setup correctly. You can correct it with the SetVNC program on the first system page , or by editing the environment variable

ENV:VNCGuide.Path

by hand using an editor of your choice. It contains just the complete path of the guide as text file.

- The icon (".info file") of the ViNCEd guide is available, i.e. a file named "ViNCEd.guide.info".

- The default tool of this icon is setup correctly. Check this by selecting the icon with the mouse, use then the "Info" menu item from the workbench. The "default tool" should contain the complete path of a program that is able to display "Amiga Guide" files, for example "Multiview".

SetVNC loads, if requested, this program and runs it to display the online help.

I don't want to keep the icon of the online help.

Solution:

If you erase the icon of the ViNCEd.guide, SetVNC will no longer be able to detect the default tool used to display the guide, so SetVNC can't be used to invoke the guide at all. However, since the help itself is requested by the "Get help" system macro on the third system page of SetVNC , you might want to adjust this macro to load the guide browser directly. For example, you could enter a command like

"Run >NIL: <NIL: Sys:Utilities/Multiview HELP:ViNCEd.guide >NIL: <NIL:"

to launch MultiView in background.

The icon I specified as iconification of the shell does not work.

Solution:

The icon path given at the fourth system page must be the complete, absolute path to the icon, without any ".info". For example, to use the standard shell icon, specify:

SYS:System/Shell

Note that there's no ".info".

The TAB expansion doesn't match files if the "Executables" priority is set to -128.

Solution:

ViNCEd regards a file as "Executable" as soon as its "e" protection bit is set. However, this bit is "traditionally" set for most files in the Amiga filing system, hence most "files" will actually match the "Executables" category and not "Files".

Iconification doesn't work any more. What's wrong?

Solution:

You run a program that requested the location of the "intuition window" in which ViNCEd outputs its text, and this program did not return the information properly. This means, since ViNCEd doesn't know whether this entity is still required or not, it can't close the window to iconify it without risking a crash by the program probably still using it.

The standard CON: handler has, by the way, the same problem: A window opened with the AUTO window path argument looses this ability as soon as the window is requested from outside.

To be able to iconify the window again, you've to release the window pointer manually by running the command SetVNC FreePointer .

This command un-does the effect of exactly one allocation of the window.

For example, each invokation of the "More" utility must be matched by one and only one call to "SetVNC Freepointer". The "More" script file supplied in this package does this automatically for you and should be used instead of running "More" directly.

The "ixemul.library" and all programs using "ixemul" are rather worse in this matter. They do not only request the window pointer once per session, but once per line read from or printed to the window, which means that a huge number of allocations accumulate. A single "SetVNC FreePointer" is truly not enough to cancel all these, but a SetVNC FreePointer All might do the job.

To big HOWEVERS:

- First, the "SetVNC FreePointer" command is not completely safe in the sense that there's no guarantee that really no program is still using the window pointer when this command is invoked. Running this command in the wrong situation may cause crashes.

- Second, even "SetVNC FreePointer All" won't help with the ixemul "pdksh" simply because this shell will print its prompt after having run the command, and will therefore request another window pointer just after you've released it. Urgh.

1.460 Thank you folks! Credits page

Special thanks goes to HiSoft for DevPac 2.0 and to the Software - Distillery for the linker BLink.

Thanks to Ernst Besser for continuously testing this proggy over years, and thanks to Oliver Spaniol for lots of useful remarks.

Thanks to Goran Mitrovic for reporting various bugs in 3.17 and some useful remarks. Not all have found their way to ViNCEd yet, but I'm working on it.

Thanks to Rodja Adolph for his useful remarks about the ViNCEd requesters. Here they are, Rodja. Hope you like them...

Thanks for Albert Bertilsson for the great idea to let ViNCEd create its own screens. This has been added to 3.20, Albert! Thanks to Christopher Naas for reporting a bug in the SetVNC program, which has been removed in the 3.09 release, and again for an enforcer hint of 3.20, which has been removed as well. To bad that I don't have a MMU. What a luck that Christopher is out there to report the hits... (-;

Special thanks goes to Nick "NLS" Sardelianos. Most of the ideas new to version 3.30 are up to him, and life will be definitely harder while he's busy with military service. Thanks a lot, Nick! You considered such a lot of options that I was unable to complete the 3.30 in time, and even now some are missing, even though the 3.60 is getting very close....

Thanks to Aristotelis Grammatikakis, another guy with a lot of useful ideas.

Thanks to Georgia Pristo, for sending DW and reporting a problem with it - plus more useful remarks.

Thanks to Steve Clark and Miles Willmek for additional ideas new ViNCEd 3.40.

Thanks to all beta testers that helped me to find bugs! It has taken a while to complete the 3.30 release, but has taken even longer without your work!

Thanks to Christopher Perver, Jochen Koob, Eric "Parsec" Spåre and Bernardo Innocenti for additional ideas for the 3.41 release and bug reports.

Thanks again to Bernardo Innocenti for continuously "annoying" me about the scroller gadgets until I finally updated the input handler. Thanks to Stefan Sommerfeld for some other hints, especially for the gadget refresh. You may now disable paths from searching - here's your will, even though I don't like it. Why don't people accept new ideas and want still this old KingKong behavour? Maybe for the same reason why they don't by PCs... :)

Thanks to Kevin A. Brown for additional ideas I added to the 3.50 release, and to Martin Gierich for his comments, his support and his fine "PatchWord" debugging tool I used for beta-testing. Thanks to Frederic Steinfels for the idea with the colored cursor I was finally able to implement.

Thanks goes to Holger Jakob for reporting quite a lot of minor bugs of the 3.50 that are now removed. Special thanks goes to Timo Kaikumaa for allowing redistribution of his UnixDirs program in the ViNCEd archive.

Thanks again to Bernie (Bernardo) for giving me the hint about the XTerm block marking bug.

Thanks to Marcus Ekelund for reporting a serious bug in the V39 and for the hard work translating ViNCEd to swedish. Cool job, Marcus!

Thanks to Massimo Tantignone for supporting customized ViNCEd gadgets by his "VisualPrefs" program and for a very helpful conversation. If you want to know more details about this customization or want to use these gadgets by yourself, consider the "Boopsis_Readme" file in the "Include" drawer.

Thanks again to Holger Jakob for reporting the AFS problem. I hope the ViNCEd workaround for this AFS bug works now.

Thanks again to Frederic Jacquet and Adam 'DC1' Polkosnik for reporting (both) the compatibility problem with "FALLBACK". That's fixed now in the 3.55.

Thanks to Peter Mattsson for the idea to open the window pre-iconified (so, not to open the window at all.... :-)

I changed the limit of the review buffer from 1024 to 4096 lines, just to please Marc Espie and his buffer requirements.

Thanks to Arto Huusko for ideas and improvements for the window "Maximize" and "Minimize" functions, Amiga+ and Amiga-. These have been reworked quite a lot.

Thanks to Wez Furlong for testing ViNCEd with the "VIM" editor; quite some bug fixes in the ViNCEd CSI parser and line manager are due to his tests. Furthermore, using the extended colors instead of bold and one of the scroll modes are due to him.

A really big "thank you" to Walter Doerwald for testing the beta-releases of ViNCEd for quite a while and for detecting lots of bugs in the early betas. Would have been harder without your help.

Thanks again to Massimo Tantignone for providing the stand-alone toolbutton image class, and for allowing me to distribute it together with ViNCEd.

Thanks to Andreas Mixich for the "PLAIN" window path argument idea. Implemented! Thanks, too, for detecting a bug in the TAB expansion routine.

Thanks to Rüdiger Kuhlmann for reporting various bugs of the early 3.6x releases, many VT-xxx incompatibilities I wasn't aware of, and other things.

Thanks to Frédéric Delacroix for his french translation and to Damir Arh for his slovenian version of ViNCEd. Further thank goes to Samppa Rönkä for the finnish translation and the ATO, and Magnus Holmgreen, for organization.

Thanks goes to Grant McDorman for reporting some additional bugs in the VT-220 emulation that have been fixed by now.

The italian translation was written by Francesco Leoni and proof-read by Francesco Mancuso, thank you for your support, folks!

Sorry to all the folks who reported bugs and I forgot to write your names down. Ooops! You're still somewhere in my EMail, but that's about 50MB of EMail I had to dig thru. Urgh! Uhm, just contact me again so I can add you here.

I do NOT want to thank Commodore for my @#%&!!! computer. (This is my 3rd computer, my 9th mouse, my 3rd fan, my 2nd SCSI controller, my 3rd disk drive, several memory problems drove me crazy, the expansion port is wrongly designed... Which idiot designed this crap? When I found out that commodore went out of business, I really ENJOYED it.)

Thomas

1.461 Version information

Now, that's quite a lot. ViNCEd is an old program, almost as old as ConMan and much older than KingKong!

1.06 First working release. Named VNC "VeryNewCon". Not all control sequences are parsed, some workaround for console.device. Finished in 1990.

1.12 and less: Many problems with cursor positioning and smart refresh windows.

1.18 Line compressor completely rewritten.

1.24 Workbench 2.04 came out. Problems with scrolling under 2.0, still no block operations.

1.25 First working 2.04, works very safe and stays unchanged for a long time.

1.28 Last 1.xx release.

2.00 Almost everything rewritten. VNC gets now a library instead of a resource.

2.01 Removed massive problems with block operations "Cut" "Copy" "Paste"

2.02 First working 2.0 release. Still somewhat beta (beta than nothing)

2.04 Problems with "Ed" and block operations. Internal Hook

2.05 Converting macros to the menu items did not work.

2.07 Marking of the line end was ugly.

2.08 Removed a lot of 1.xx trash.

2.09 Workaround for console.device bug implemented, Form Feed corrected magic Help key.

2.10 Improved the LibInit procedure with useful alerts (-:

2.11 Removed ScreenToFront bug, new AutoSize SetVNC.

2.12 ConClip OpenCount, smart mount, NOICONS bug removed.

2.13 Menus get the right font, SetVNC keeps track of the title bar, removed again some 1.xx trash.

2.14 Added FindCNWindow(), now used in SetVNC

2.15 Priority of semaphores corrected, menu size adjusted for tiny fonts.

2.16 Added Ctrl+Cursor keys.

2.17 Menu functions now BOOL instead of void.

2.18 DOS commands now send thru owner instead global port. Lot of updates necessary, removed several bugs (sigh!). Break and owner handling now more dynamic, workaround for "more" (close event expected without request).

2.19 Again, handling of the close gadget in raw mode updated.

2.20 Removed bug in erasing complete lines and bug in prop gadget procedures.

2.21 Scroller get disabled; added another workaround: If the DOS moves the cursor, user inputs are valid starting from this cursor position. This simplifies the input of tables, and works somehow like the screen editor of the Atari 800 XL.

2.22 Scroller disabling now much better, again corrected owner handling.

2.23 Lot of problems with owners, gofer does not work (WaitDosPacket is broken). Rewrote owner system, private hash list for owners. Several days of work. Found bug in rexxsyslib (WaitDosPacket).

2.24 And again owner problems with programs that do not open own streams. Sigh. Corrected read & write. Will this stuff work now?

2.25 I knew it: Corrected owner handling once again. Should not make problems again, but who knows? Again removed some obsolete procedures of release 1.xx beta testing.

2.26 Added FilterInput, HandleKeys updated, and Hook procedure. HandleKeys modified for TABHook. OwnerSemaphore added.

2.27 VNCClip.supervisor added for iff clips. ConClip is now obsolete. Had lot of trouble with the library, found some bugs, esp. with memory problems. vnc.library modified again, removed another (old?) bug in internal Copy/Paste. TABHook stack enlarged, VNC needs at least 4000 bytes, esp. if you run it with a debugger.

2.28 Trouble with ActivateWindow under Kick 1.2/1.3. Does not seem to work better due to a bug in the 1.x libs. Again updated the TABHook to make it working with a NULL lock (Now version 1.01). Forgot to remove option from Makefile, which leaded to illegal hunk linkage.

2.30 Support for locale.library. Prefs are now loaded by the supervisor, added OldOs flag. Removed bug in KillSuper, and unnecessary BSS segments. BuildMenuStructs bug removed, can only seen with "MagicMenus". Again problems with the f*cky owners, removed bug in AClose. When will this sh*t work finally?

2.31 Minimal changes in the lib, new hook for AppWindow. Window size depends now on the text overscan. But completely rewrote SetVNC (2.11). Help now by AmigaGuide, own localization, TABHook program is now obsolete and integrated into SetVNC. New feature: VNC is now an AppWindow (icons welcome).

2.32 Added ALT open option, removed bugs in the parsing of double quotes. (Leftover from 1.xx?) VNCNewWindow now own structure. LibInit and LibRemove remain now in forbidden state, all asynchronous processes are done by the supervisor. Failed to close timer device. This release works pretty well now.

2.33 Removed minor bug in FindCNWindow. Could return TRUE if window was not a VNC window. Caused a crash of SetVNC if called thru a AUX: window. Calculation of cursor position wrong due to rounding errors of divisions.

2.34 Concealed mode handling updated - UserType/UserPenPair can't be controlled from DOS any more - you have to use the structure, like it should. SetVNC's TABHook unstands now double quotes and pathes with spaces. Support for directory assigns added.

2.34/2 No new VNC release, but added the double TAB requester to the TABHook code in SetVNC, which is now in release 2.15.

2.35 Support for NewLook menus of workbench 3.0 added, and removed two tiny bugs. The compatibility workaround to console.device had an illegal pointer, and another in the main code which only occurred if the blitter was too slow. Added support for negative width and height in the open path, window flags are now allowed in every position, to make ARexx working. Sort of dump to place flags there, but want to stay compatible, so this is really not my problem. Rewrote routine to start AmigaGuide, had trouble with MultiView (needed PROGDIR:). I hate this program, it is very likely to crash, esp. with animations. Removed another bug in the DOS module: WAIT/AUTO windows that never opened actually got never closed.

2.36 Scrolling got faster, scrolled only used bitplanes like CON: in 3.1. Lucky not too much trouble. Again removed a problem with window activation under 1.2/1.3 OS.

2.37 The pattern of the inactive cursor is now static and no longer in the TmpRas. This prevents some misprints while scrolling of obscured window parts. Updated the edit buffer hack for table entering, parse position gets updated if one erases or adds characters.

2.38 And again trouble with owners. Action_SetPort refused to accept a port which it could not find a owner for. Starting with this release, the official owner who opened the window gets this port. Not nice, but what to do? The problem is, that you may share streams in AmigaOs.

2.39 Internal beta

2.40 Added rebuild delay.

2.41 Trouble with rebuild delay and "more", added scroll delay. Last 2.xx release.

3.00 Finally started working with 3.xx. Rewrote complete parts of code, esp. owner handling. This was too messy anyhow. Added REAL job control to replace this mess. The TABHook gets now an integral part of the library, called thru a vector. Added button gadgets, and nice tiny arrows at the scrollers. Added VNCFiler.supervisor, displays requester. Removed multi directory assign support, this should be part of the DOS and not of every program. Will write a patch later on. The job control thru Ctrl-Z looks really like a bad hack, but works astonishingly stable.

3.01 Added a lot of CSI sequences I found in XTerm, xwinshell, VT-220 and much more. Still a bug in the TAB expansion, does not find the right argument in the command line. I really thought I tested this... Added a workaround for a bug in csh, sends an illegal CSI sequence and expects the right answer...

3.02 Added support for borders. A lot of testing must be done, and all the scrolling procedures must be rewritten.

3.03 Found a bug in one of the line insertion routines. I fix this one now for the 3rd (4th,5th ?) time and it still does not work. Ugly.

3.10 Thought an iconify gadget could be a neat idea and added it. Lots of work! Found a bug in the shutdown procedure of the VNCFiler. sigh! Completely rewrote SetVNC, old code was a mess after five years of fixing.

3.12 Again some bugs in iconification. Added Job control to SetVNC, really forgot I had to.

3.13 Help menu item did not work, and SetVNC did not save the guide path. I though this worked, but... Added "fork" script and the ability to suspend owners, and removed a bug from Foreground and Background lib functions. Forgot to count the NUL.

3.14 Job control in SetVNC updated, using now "other" for hard cases. Probably should add some more options. Foreground and Background library functions now got flags, too good I hadn't made this public. Added again two hacks for compatibility with "Ed". It sends illegal CSI sequences not compatible with VT-220 and XTerm. Removed crashing "Ed", leaves port with unanswered Read request, I have to flush this manually. Thought I had this trouble once before in one of the 2.xx releases and removed the patch cause it looks so ugly. Updated testing of foreign menus. I wonder why intuition did not crash?

3.15 Thought with 3.xx the owner handling finally works. Proved again that I was wrong! ARRGHH! Fixing the "Ed"-problem added another bug. Really silly, but I think THIS TIME I really made it... (hopefully).

3.16 Tiny bug removed from the window closing procedure. Switching the window to RAW: mode canceled the AUTO and WAIT state completely and made it impossible to close such a window. Removed some bugs in job control: Can't send ARexx to background, crashed when disabling a stopped process and some more. Found a bug in a "correction" of a cursor movement procedure. Sigh. Fixed another bug: Graphic mask was calculated wrong if a single character has to be marked in an empty line.

3.17 TabHook expansions are now sorted by type: Devices prior directories prior files. Thanks, Olli, for the tip! The fix was quite easy and done in five minutes.

3.18 Fixed a tiny bug in the Prefs correction routine. An empty macro does no longer end the parsing of macros, but is accepted as a valid entry. Found a tiny bug in the MatchFirst() function and added a workaround. Fixed a bug in the SetVNC program that might cause a DeadLock on few machines. Replaced the arrow gadgets and scrollers with Boopsi images, if available. ViNCEd looks now O.K. with the new 3D look. Tested with SysIHack, SysI2,MCP, Urouhack and works. However, no standard iconify gadget yet. Removed some compatibility hacks for the console.device, since it does not properly ignore unknown command sequences. Iconify gadget is no longer installed if there's not enough room in the title bar. Fixed two additional bugs in the Prefs correction procedure, and removed the creation of a lock to the window on invocation of a macro. Added an additional check in SetVNC to prevent a crash if a GadgetUp message from a non-SetVNC gadget is received. The prefs flags did not set the gadget properties correctly on installation of the prefs. Sigh!

3.19 Fixed a lot of bugs in the block operations with borders activated. Looks like I forgot these to update in 3.00! Fixed another tiny bug in the line compressor code. I added a "have to think about it" mark to it in 3.00, but I never did. Sigh! Added two options for file requesters and a lot of support stuff in the dos interface module. Had a lot of trouble to avoid deadlocks between the VNCFiler and the dos handler and finally found some use of the pr_WaitPkt pointer. Added a new cursor control mode, XTerm mode, since some users requested it. Removed a bug in the set border routine that might cause a harmless guru - ViNCEd waits now until intuition resized the window. This works only in 2.00 and up, using a delay for 1.xx. Removed an unnecessary flicker in this routine as well. Added some options to the menu, and removed some never used. Made the "Smart Close" even smarter.

3.20 Added support for private screens which can be build on request, and fixed a bug that caused SetVNC to crash if no vnc.library was present. Fixed one bug in the guide about the WINDOW path argument.

3.21 to 3.23 These have been beta-releases. They were never available thru AmiNet. One enforcer hit while scrolling should be gone now. The open path argument parsing was a bit buggy and caused compatibility problems with various programs. The "Next Screen to Front" menu item caused hangs with some additional software. The ConMan "L" option was broken and turned on the seven bit mode accidentally. The TAB expansion routine was a bit buggy with expansion of an empty string, and expanded wrongly if the cursor was placed behind the string. The ordering of the "ICONIFY" open path argument was relevant. The ViNCEd TAB expansion requester did not allow to enter directories. The Boopsi routines were buggy and wasted CPU time. Cursor position selection with the mouse locked the RAW window emulation. As you see, a lot of bugs have been found.

3.30 The new final release. New stuff added to this and the previous beta releases: Window title control codes, new CSI sequences , a better ViNCEd expansion requester , better custom screen support by more open path arguments, icon drop qualifiers, default screen mode and font support. A lot of bugs have been removed that I added to the beta releases. The incompatibilities with DW, MWM and other programs are gone - I hope completely. A flag was added to search only the most useful part of the path in TAB expanding file names, plus some other features. Added the "KEEP" path option, updated the dynamic memory manager - faster and uses less memory now. The guide location is now kept in an environment variable as well.

3.31 Improved the ViNCEd requester once more. There's now no reason to use the standard requester any more. Added a hack to allow VirusWorkshop to check the ViNCEd archive without trouble - manual installation is now no longer possible, due to the encoding of SetVNC. Be warned!

3.32 The BOOPSI scrollers got not updated if the number of visible lines changed. This bug was hidden by another bug in earlier releases. Sigh.

3.33 Internal beta: Found another bug leftover from 2.xx in the line compressor. The settings menu item "Expand with TAB" did not work properly. Added the private CSI sequence "CSI SPC s" for CON: compatibility and ANSI colors. Enhanced the window sizing algorithm to make smaller windows possible. Added asynchronous type-ahead, more flags to control the prop-gadgets, better scrolling, hard cursor scrolling stop, path keyboard control sequences, more flags for TAB expansion, shrunk the minimal size of a window and added another check for the window size, updated jump scrolling, changed the order of the buttons. Added numerical key pad cursor control. Again a lot of work was done!

3.40 Removed an enforcer hit of the 3.33beta2 and one duplicate entry in the library functions. Removed another tiny bug in the boopsi handling.

3.41 The "ESC c" control sequence reset the ANSI mode to the settings stored in the prefs, not to the mode set in the window title - removed. The set render command for color nine did not work. Keypad-mode and Alt-movement conflicted. Added CSI sequences and flags to the prefs to set the ANSI coloring to something different than default. The complete color addressing has changed a bit. As a side effect, screen colors set with CSI V WILL re-appear if the window gets re-opened after an iconification. The old 3.40 dropped these colors (unlike what it should do). Fixed another bug with gfx boards: The color of marked blocks was different at marking and re-printing time, due to the uncertain behavour of the COMPLEMENT drawing mode for gfx boards. If the screen mode is set to CHUNKY, ViNCEd does no longer try to speedup the block marks with COMPLEMENT drawing, but reprints them. This might be a bit slower, though. Removed a bug in the raster mask calculation routine that might have caused graphics trash for marked and scrolled line feeds. Added a flag to keep the bottom of the window always aligned to the bottommost line in the display buffer. Added a flag to ignore icon ".info" files in the TAB expansion unless you ask for them explicitly. The TAB-Expansion cursor placement policy changed a bit, I think to the better. Added a flag to exchange Shift+ALT Del/BS with ALT Del/BS. ViNCEd installs now a no-op backfill layer hook if possible, hence giving a faster and lesser flickering refresh. Added another "Edit" and "Shell" page to SetVNC to keep the new flags introduced to 3.41. Added the "NumL" key function.

3.42 Removed a bug in the gadget refresh. Forgot to handle this correctly if the prefs get updated. Inserted additional explicit gadget refreshes. The ViNCEd input handler reads now mouse move events if enough CPU power is available, resulting in much smoother scrolling. Fixed a tiny feature of the WaitForChar() function. It did not operate like it should if an EOF signal is pending. Fixed! The guide contained a bug :) The explanation why TAB expansion in kshells did not work was wrong. Sigh. Removed a problem from one of the refresh routines that failed to operate with the no-op backfill hook introduced in 3.41. Added the "harder soft reset" with the Shift key hold down.

3.43 The CloseScreen routine for public screens was broken - ViNCEd was from time to time unable to close its own public screen. Added the "Jump to Next Screen" menu item and a flag to disable the C: path searching at all. The TAB expansion flags parsing was a bit messed up. Everything worked, but the flags had an interpretation different from what the guide said. This is fixed now. The font loader was buggy - called the ASCII string to binary converter with a wrong argument. Fixed.

3.44 This one was a public beta release. I rewrote the clipboard handling completely, added the colored cursor and the TAB expansion. Another new feature are the buffer I/O related functions in the project menu.

3.45 Another public beta. Removed quite a lot of bugs of the 3.44, and just another two bugs left over from 3.43: The line allocation routine had a major problem (urgh!) and the color arbitration logic was somehow messed up.

3.50 Final checks: Added another string to the localization, removed a bug in the sizing of the scrollers (problems with MCP now fixed). Removed a bug in the 1.3 support procedure. Changed the prefs style saving/loading that seemed to be unclear to certain people, added another flag to disable ViNCEd's feature to lock the window on a mouse click. VNC: is now a legal path except for the antique 1.2 and 1.3 versions of the Os due to a bug that used to be in the Mount command.

3.51 The history search is now case-insensitive, I guess it's more useful this way. Made also some minor changes in these routines. Another change is that a wrapped-around in the TAB expansion list inserts now a blank argument, to inform you about what has happened. Fixed a bug in the XTerm mode support - the arrow gadgets near the scrollbar did not work like they should. Softlinks are now supported by the TAB expansion logic. Removed a bug in the window deactivation and in the block marking procedure. ViNCEd tries now to abort a running ExAll() call in the TAB expansion. Removed some mess in the insertion routine threatment of the marked line end. Removed a bug in the block marking routine that caused sometimes a line to be inserted on top of the screen. Removed a bug in the color arbitration routine that allowed identical colors for complemented colors. Updated a part of the TAB expansion routine - it corrects now the case of the device and directory names.

3.52 Removed a bug in the TAB expansion that refused to work with "/" and ":" directories. Added the "RIGIDCURSOR" flag. Removed a bug I added by accident to the 3.51 and another refresh related feature. Scrolling while marking the text was impossible before, fixed.

3.53 Added the duplicates history flag, changed the lock management when dragging blocks. Added forward/backward arrows in the Prefs editor and the IfVNC shell argument. The window gets now activated if an icon gets dropped on top of it. Changed again the TAB expansion a bit: If a unique device or directory is found, the double TAB requester will directly go into it. Changed the full screen refresh a bit, I hope to the better. Should be more effective now. Fixed a bug in the Boopsi-Iconify support - VisualPrefs works fine now and added even more Boopsi-fication support. Added a workaround for a serious bug in the V39 ExAll/ExAllEnd() routines which supports ExAllEnd() for V37, too! Renamed the "iconifyimageclass" to "tbiclass". Check the "Boopsis_Readme" in the "Include" directory.

3.54 Updated the ExAll() workaround because it conflicted with a bug in AFS. The replacement code for the buggy V39 ExAll() ROM code was actually taken from SetPatch, but it turned out that this implementation caused problems with AFS; the reason is a bug in the AFS Examine() and ExNext() handling - sigh. Thanks, Holger, for reporting. Rewrote the communications routines between the TABHook and the handler process. The old implementation was a strange mixture of synchronous and asynchronous message passing that caused a lot of problems. The new implementation is a lot less messy.

3.55 ViNCEd can now be mounted as RAW console handler. A "SetVNC Mount Override as RAW" will work fine now. Mountlist users might set the "Startup" argument to "1" to get a raw window. Added the "ICONIFIED" window flag. Changed the default from "NOFALLBACK" to "FALLBACK" to prevent problems with certain programs. Added the "NOFALLBACK" option. Removed two minor bugs in the window open routine (which is by far too big anyways due to the mess of "optional options"...) Added another cursor shape that is used when the output is locked, used by ˆS.

3.56 Added the ˆR and ˆB keyboard commands, fixed a bug in the "ViNCEd TAB expansion" requester, added "*" as possible screen name to refer to the frontmost public screen. Added a kludge to make the ARexx command LINES() working. (Messy, messy!) The history functions work now a bit different. They don't longer move the cursor any more. Might be useful for ˆR.

3.57 ViNCEd ignored multiassigns to the command directory C:. ViNCEd locks now the layer to get the window dimensions. The "Close Window" RAW event is no longer included if you open a ViNCEd window in RAW: mode. The ACTION_SET_FILE_SIZE packet returns now the proper return value (-1 instead of 0), the mn_ReplyPort field of the packets send to ViNCEd is no longer trashed. Improved parts of the packet handling code on the way. Added another bunch of packets, for example AREXX compatible ACTION_DROP. ViNCEd does no longer use fh_Arg2 for identification and leaves that field blank. ACTION_SET_COMMENT can be used to set the window title now, just like ConMan (Try 'Filenote CONSOLE: "A nice feature"'). Removed two bugs in the suspend (ˆZ) feature, one minor, one serious... Urgh!

3.58 Removed another bug in the TAB expansion requester - forgot to save back one register. This might have caused crashes in certain situations. Changed the format of the version string slightly, to avoid problems. ViNCEd will again install itself as an AppWindow in public screens, to allow icon drop into the window if started from DOpus.

3.59 Removed a bug in one of the TAB expansion routines that caused a conflict with SnoopDos.

3.59a A release that was never published. It just increased the stack size of the TAB expansion to avoid conflicts with certain patches.

3.60 Uhoh, writing all changes down would really break this chapter. So, for short: New Features: ASCII preferences, configurable keyboard, configurable TAB expansion functions (six or 24, up to how you count), including a set of priorities and flags for each function. More CSI sequences, more flags. Huh, lots of bug fixes of bugs in the 3.59 version, as for example errors in the line manager, the character insertion routine, the block marking routine, misinterpretation of some CSI sequences. More options on the window path, added window default title in the preferences, fixed bugs in the interpretation of printed backspace, added more control sequences for the window title, the location of the preferences file was changed, bugs in the icon drop routine. Included a new and reworked guide (this one), included a FAQ, a new index. Most of the nodes have been rewritten, one of the reasons for the more than half a year of delay.

3.61 Added more support for Massimo's standalone toolbutton image class. ViNCEd will now try to open the "titlebar.image" and use this if it is available. This image class will customize ViNCEd's macro buttons as well as the iconfication gadget in the title bar. The class itself IS NOT required if you're running VisualPrefs anyhow. More on how this works in Massimo's docs. Added the "PLAIN" window path argument, idea by Andreas Mixich; thanks!

3.62 Fixed a bug in the TAB expansion routine that used the wrong definitions, thanks again to Andreas Mixich. Changed the Ctrl-C break logic a bit, it looks now like the original CON: method, even though that doesn't make me happy. Fixed a possible bug in the SetVNC program that might have crashed if the guide file is shut down. Added "Save To" and "Load From" gadgets to the new second "About" page of SetVNC.

3.63 All keyboard sequences use now really the seven bit variant ESC [instead of CSI if the seven bit mode is enabled. "Toggle ESC" does no longer answer the ViNCEd "Toggle ESC" sequence in raw mode, but produces a literal ESC 0x1B in the output stream. However, in order to simplify the parser of additional programs, the ENGLISH mode will always send complete CSI or

ESC sequences, therefore a program will hear "Toggle ESC", "Insert ESC" or "Insert CSI" keyboard sequences directly in the ENGLISH mode. ViNCEd does now keep a separate flag whether a line was broken on the right window edge or not and will not insert a line feed if these lines are copied to the clipboard. The sequences reported for ˆC thru ˆF were incorrect in RAW mode with the "direct report" flag enabled, fixed. The line wrapping algorithm for VT-xxx was broken. CSI E and CSI F did not take arguments. Cursor movement ignored scroll lock and word wrap disable. Backspace wrapped around in VT-220 mode. ICONIFIED open path flag did not work due to a "bug fix" in one of the beta versions. Added a new flag >?15h to restrict reported sequences to 7 bit modes. Thanks goes for a lot of that to Rüdiger! Great! Updated, too, the SetVNC once again. This time, the second keyboard page was reworked to be able to browse the keyboard bindings made so far. To see the current keyboard binding, go to the second page, select a function from the list and use the "« Prev Key" and "Next Key »" gadgets to see all keyboard sequences bound to the selected function. Incremented the SetVNC version to V41, for consitency with the main library.

3.64 Fixed only a minor glinch with the scroll lock flag and gadget activation. Thanks to Holger Jackob for digging this out. Except that, and including the up-to-date locale sources, nothing needs to be changed.

3.65 Directories of MS-DOS type filing systems are no longer cached because CrossDos doesn't update the volume date correctly. Included the NamedConsoleHandler which offers a workaround for a feature in the pre-3.0 Os. Ctrl-Z works now for Os 2.x, too, provided this handler is installed. Added the "BACK" window open path option, lowered the the minimal timing values. Because again nobody seem to read this guide, the "Raw control" bit CSI >? 26h defaults now to "on", i.e. all control characters are send literally in RAW mode and are not executed. As an additional hack, ˆC to ˆF execute always in CBM compatibility mode, indentically to what RAW: does. This *does not* go for VT-220 mode. Added code for a window-restore function, it is however not yet available in the menu, this will be done in 3.66. Added a special check in the startup code to avoid possible deadlocks in case ViNCEd is started and ENV: is not yet mounted. The ˆQ and ˆG control sequences were buggy and not sent correctly. ˆQ sent erraneously a ˆS and ˆG did always ring the bell and never sent any code. Argh! Fixed the documentation, the "send window reports" sequence is actually "CSI 0 SPC q", not "CSI 0 q". This doesn't make any difference for ViNCEd, but the console.device recognizes only the former sequence, not the later. Thanks to Gunther Nikl for reporting. Added the "CSI >?23l" sequence that disables the emulation of a RAW: bug whose emulation added for completeness in 3.65. Shudder! Included a finnish translation, thank goes to Samppa Rönkä.

3.66 Added the "Restore Window" menu item. Fixed the broken "Insert CSI" and "Insert ESC" sequences of 3.65. Added the italian translation by Francesco Leoni and Francesco Mancuso, thank you! Added mouse-wheel support for serial mice, the wheel scrolls the window up and down.

3.67 Added the %l and %o window title command to insert the state of the NumLock and Overwrite qualifier. ViNCEd will now try to adjust its icons correctly if some other program hacked more than the system default icons into its title ("PowerWindows"). The shortcuts of the ViNCEd windows can now be localized. Fixed a bug in the TAB expansion cache handling of multiassigns. Added a workaround for a bug in VIM.

Read also the credits section of this guide, to find out who is responsible for reporting the bugs (-;

1.462 Future Plans with ViNCEd

No, the story isn't over yet. Here's a collection of future plans with ViNCEd and related:

Plans for 3.70:

- Remove 1.2 and 1.3 compatibility completely. Could be one of the rare releases where ViNCEd gets actually smaller.

- Add "smooth scroll"?

Plans for 3.80:

- Configurable menus?

- Unix style "screens" support?

Plans for 4.00:

- Complete rewrite in C, port to PPC architecture.

- Rework the ViNCEd concept. Start with a VT-220 "widget" (boopsi) for the CSI parser and the editor features. On top of that a "vinced.device", on top of that a "ViNCEd-Handler", on top of that "Vinchy", the ViNCEd shell.

1.463 Bug notes and reports, how to contact the author

What, do you expect ViNCEd is bug free? Gee, to give you an impression how big this thing is:

The assembler source (!) of ViNCEd and SetVNC is about 2.4 MB in size, not counting the includes, this guide, the Makefiles, the Agenda, the includes, the autodocs, the fd-files, the installer script and the "Extras". Do you think I'm crazy to write this in assembly language? Guess you're right....

How to report me in case of a bug?

Please include the following information:

o) Your name and an address how to contact you. EMail preferred.

o) The version of ViNCEd you're using.

o) The version of the Kickstart and Workbench you're using.

o) Other patches active at that time.

o) How to reproduce the bug.

o) If possible, run the Enforcer. Configure it in a way such that the stack of the crashing/buggy procedure gets dumped, too. The more lines, the better. This stack dump helps usually quite a lot.

o) Run the SegTracker, and MungWall, and PatchWork; then try to reproduce the bug. Send me the output of all these programs in case of a crash/bug.

o) Include a copy of the ViNCEd settings active at the time the crash happened. They can be usually found in ENV:ViNCEd.prefs. It's a plain ASCII file.

o) In case this crash/bug happened in relation to another program and this program or a demo version of this program is freely available, include information where to get it.

Ship all these information to my address .

Thanks for your help!

So, here are some features of ViNCEd I haven't been able to reproduce on my machine and that haven't been fixed for that reason. In case you know anything specific about these, please lemme know:

Icon drop support seems to make trouble for some people. If you open a ViNCEd window before the workbench is open, ViNCEd won't be able to install its icon drop feature into that window. However, all windows opened AFTER the workbench pops up *should* provide icon drop, except the windows already open before. This works fine here on my system, and I haven't been able to reproduce the problem. You may re-enable icon drop support by restarting the "TABHook" with the CSI-sequences CSI >?12l CSI >?12h.

There seems to be a problem with the MudPad math program. This program is no longer available and hence I haven't been able to reproduce this bug. However, this problem might be related to the ViNCEd settings. You should not set the Inhibit Horizontal Scrolling flag when running programs in RAW mode, or should at least enable the Line break at right border as well if you must use this flag. Not following these rules might cause the cursor leave the viewable part of the window without causing a scroll.

ViNCEd seem to cause crashes if it gets iconified while some output is in progress. Type "dir dh0: all", then iconify the window. Causes either a mungwall hit, or a crash/hang. Does not happen on my system, tried with various configurations without success. Analyzed the critical code several times without finding anything suspicious. If anybody knows how to reproduce this, please let me know. Possibly some strange hack that is active? (Might be related to some old versions of ModePro, should be fixed with later releases.)

ViNCEd seems to cause a mungwall hit if the window gets iconified on a non- native screen. The mungwall hit is in the intuition DisposeObject() routine (the precise address depends on the ROM version) which seems to deallocate zero bytes of storage. I stepped thru the parts of ViNCEd that allocate and release boopsis, without finding anything suspicious and without being able to reproduce this bug. Probably a CyberGfx problem? Probably the same ModePro bug as above?

ViNCEd seems somehow to interact with the AmiIRC program and ARexx scripts for AmiIRC. Using ViNCEd and AmiIRC together seems to break some arexx scripts, even though no ViNCEd window has been opened at that time. This problem is still a mystery for me. I can neither reproduce it, nor have any idea what the reason for this could be. Would be interesting if the ARexx patch included in the full distribution - which fixes an internal ARexx message queuing problem - can correct this bug. This might be, too, a timing problem of various "hacked" and critical ARexx scripts.

1.464 The tbiclass Boopsi Interface

ViNCEd uses the system "Boopsis" wherever possible; these can be replaced by a "SysIHack" style program like "VisualPrefs". However, since not all gadgets required by ViNCEd are available as public gadget classes, an interface has been worked out together with Massimo Tantignone, the author of VisualPrefs, to allow replacement of the remaining ViNCEd gadgets. This interface works thru a custom gadget class, named "tbiclass", short for "tool button image class". ViNCEd will first try to allocate boopsis of this class and falls back to its own gadgets in case this class is not available.

The "tool button image class" is either provided by a stand-alone boopsi which was written by Massimo Tantignone and is included with his friendly permission in this distribution, or is installed and maintained by the VisualPrefs program of the same author. Hence, if you're running VisualPrefs anyhow, there's no need to install this image class.

If you want to write a "GUI improver" software....

...you should supply a public "Boopsi Classes" called "tbiclass". The class should be a subclass of the ordinary "image" class and will be allocated by ViNCEd for the iconify gadget and the "title bar" gadgets. The image class should come in at least two "flavours", determinated with the "SYSIA_Which" tag. This tag is defined in "intuition/imageclass.h".

Details about the iconify image flavour of the "tbiclass"

This image subclass is used to render the iconify gadget in ViNCEd and other windows.

You receive the following information thru the standard tag items:

SYSIA_Which with a tag value of 104, defining the "iconify flavour".

o) Width & height of the image to create. These values will be the dimensions of the "depth arrangement" gadgets in the window. You may well ignore these parameters as long as the specified dimensions differ by the returned dimensions by a reasonable amount - meaning: screen sized gadgets won't work, but everything that fits in the titlebar should. ViNCEd tries to adjust its GUI accordingly to make enough room for your image.

o) The DrawInfo of the screen the window is to be created on. You should use it to choose the correct pens for your image.

ViNCEd will allocate only ONE instance of this boopsi class and use it for both, the regular and the hilited image of the iconify gadget. Make sure your boopsi class is smart enough to render itself accordingly, you receive all necessary information thru the standard boopsi "message".

You WILL NOT receive a recessed/raised flag of the image to create. As I said, your boopsi class is expected to be smart enough.

To be compatible to an intuition "feature", ViNCEd expects that the LeftEdge of your returned image is set to "-1", hence, the image is shifted by one pixel to the left relatively to the gadget. This sounds a bit strange, but intuition treads the standard system images in quite the same way. The reason for this displacement of the image is to keep the left black image border out of the sensitive area of the gadget. This dark border appears later on to be the apparent right border of the title bar.

Details about the button flavour of the "tbiclass"

This image subclass is used to render the "titlebar buttons" of ViNCEd windows.

You receive the following tags:

SYSIA_Which with a tag value of 106, defining the "button flavour".

o) Width & height of the image to create. You may well ignore the height parameter as long as you make sure your image will be the same size as the title bar of the window. You ABSOLUTELY MUST obey the width parameter, ViNCEd relies on the correct value set in the boopsi returned.

o) The DrawInfo of the screen for rendering informations.

You WILL NOT receive a recessed/raised flag for the image to create. As above, ViNCEd expects that your image class is smart enough to render itself accordingly, all necessary data can be received from intuition.

ViNCEd will allocate only ONE instance of this boopsi per button, and will use it for both, the gadget image and the hilite image - the pointer goes to the "GadgetRender" and "SelectRender" fields of the gadgets.

Expect that parts of your images will be obscured by text, the contents of the button.

To be compatible with a feature of intuition, ViNCEd expects that the image is shifted to the left by one pixel, i.e. the LeftEdge entry of the image structure must be set to -1. See above for details.

If you write your own application and need an iconify gadget or a title bar button...

First, provide your own standard gadget imagery. Make sure your program works on a plain system, without any "GUI improver" software. If you want your gadgets to get standard images, try the following to get a standard iconify gadget:

Allocate a boopsi of the public "tbiclass" class, provide the following tags:

SYSIA_Which with a tag value of 104, defining the "iconify flavour".

o) Width & height, and the DrawInfo of the screen your window is located on.

If this allocation fails, fall back to your own image.

If it works:

Check the width and the height of the returned image. They might be different from your specifications in order to give a "better look". Adjust your GUI accordingly, make some room for this image. Fill the pointer you received in the "GadgetRender" and "SelectRender" fields of the gadget structure and specify alternate images as gadget rendering mechanisms. MAKE SURE YOU ADJUST THE GADGET SIZE to reflect the dimensions of the image!

To be compatible to an intuition feature, the image you receive will be shifted to one pixel to the left. DO NOT ALTER the image structure you received, it isn't yours! Move the gadget instead if you need to do so! Read the paragraph above for more details.

If you need a standard button in your title bar:

Allocate a boopsi of the "tbiclass", provide the following tags:

SYSIA_Which with a tag value of 106, defining the "button flavour".

o) Width & height of the button you need. The "width" can be as big as you like, but the height should be the height of the title bar of the window. If the window is not yet open, specify a reasonable value, it will be adjusted for you. DrawInfo of the screen your window is located on. THIS TAG is ABSOLUTELY NEEDED!

You receive either NULL if the class isn't available. Fall back to your own image or border if you get this as result. If you don't get NULL, you're holding one instance of an image-like boopsi. It's width is guaranteed to be the width you specified, but the height might differ to adjust the image in the title bar. Set the "GadgetRender" and "SelectRender" fields of an intuition gadget to this pointer and specify images as regular and selected rendering of the gadget. MAKE SURE YOU ADJUST THE GADGET HEIGHT to reflect possible changes.

To be compatible with an intuition feature, this image will be shifted leftwards by one pixel. This will move the 3D-look border of it out of the gadget activation region and is an intuition feature. Please DO NOT adjust the image structure yourself, it isn't yours. Move the gadget instead if you have to.

For further questions, contact me .

1.465 Index