Index:

1. A not so little introduction:

1-1. Word abbreviations:

CShift -> 'Caps Shift' Spectrum's key.
SShift → 'Symbol Shift' Spectrum's key.
Char (o chars)-> Character/s.
Charset → Character set.

1-2. User interface (View y Edit Mode):

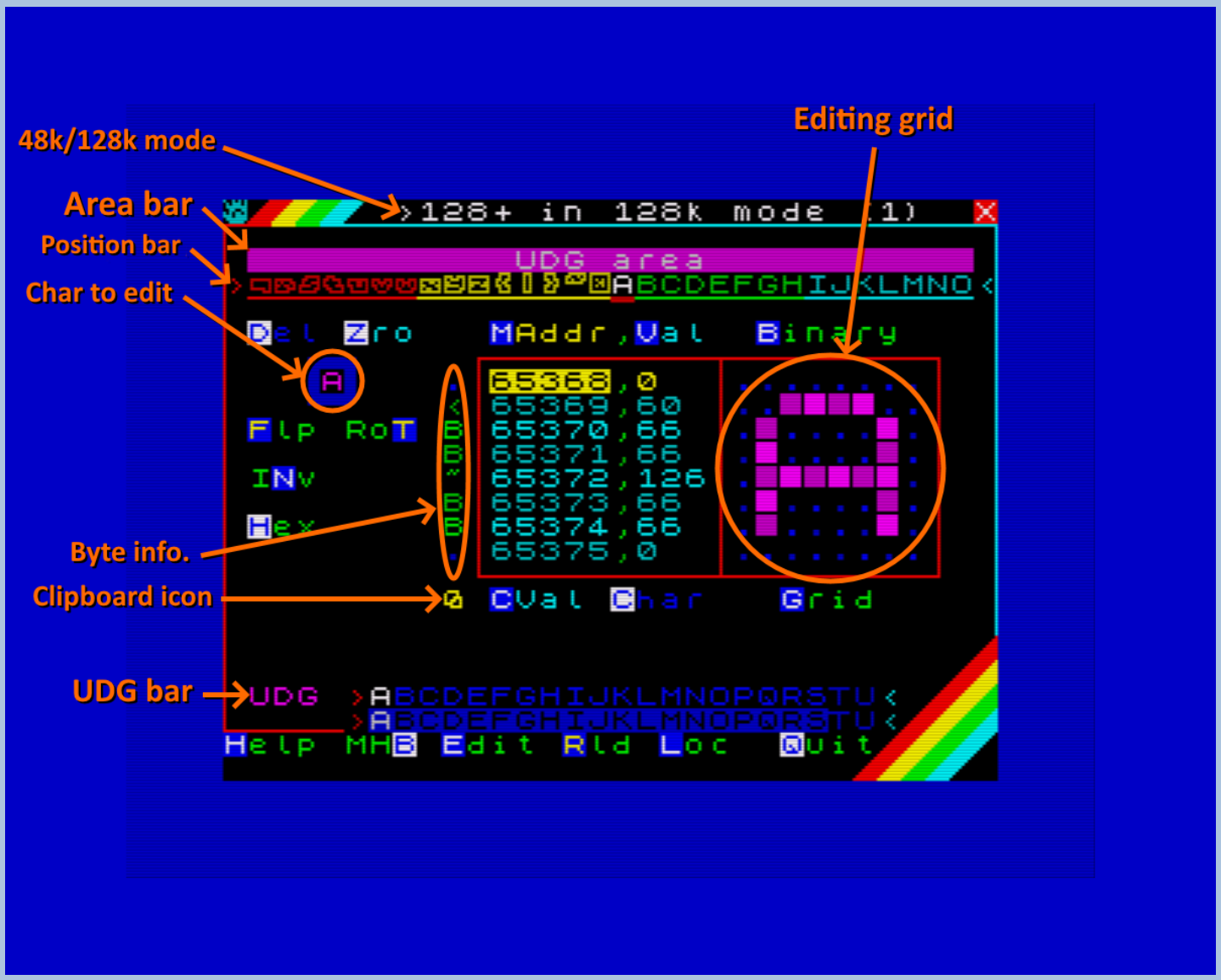


Image 1 – *View Mode.*

– Introduction. Jargon of words, User Interface –

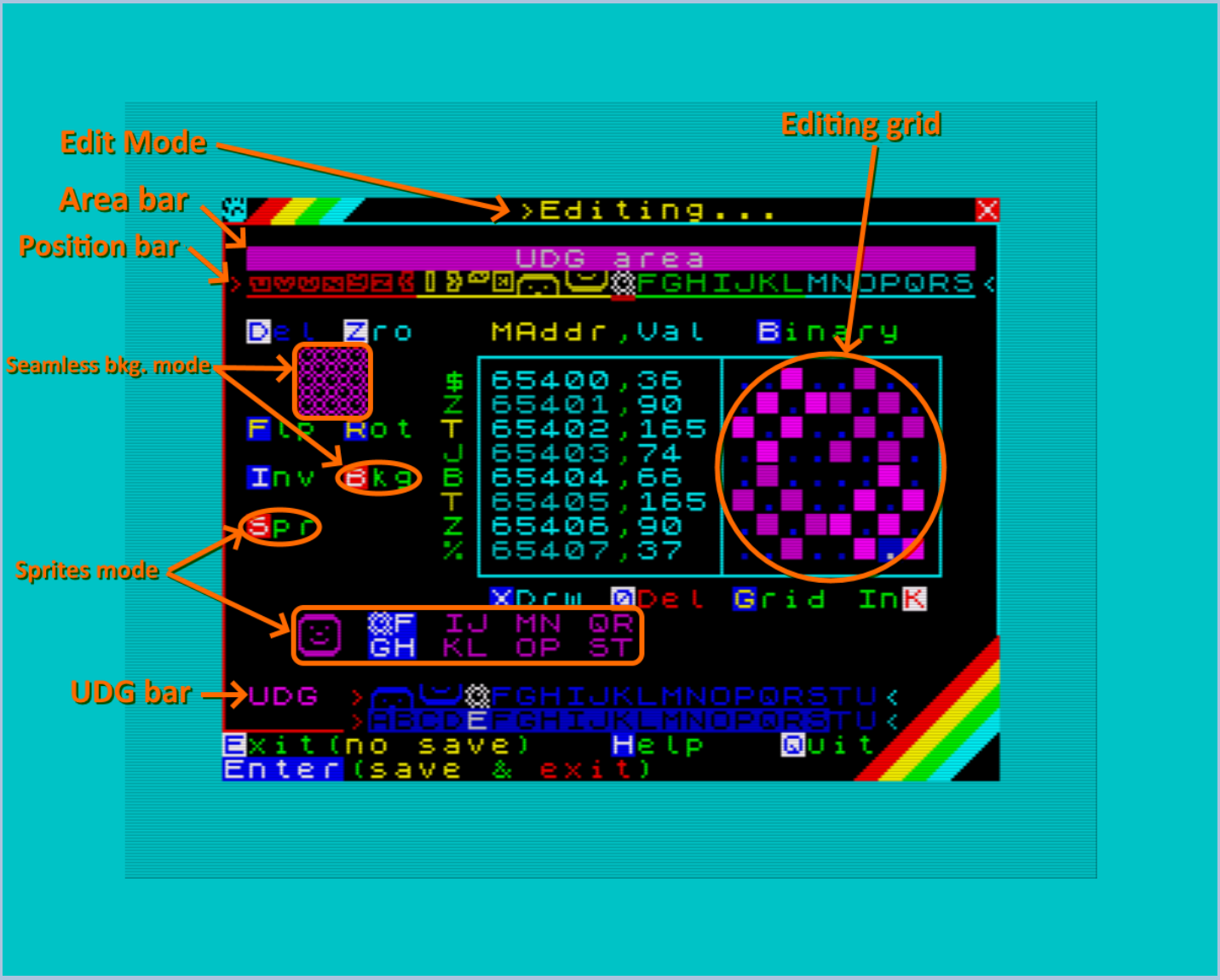


Image 2 – *Edit Mode.*

Use this images to become familiar with the interface and its various sections and names.

1-3. What is View Mode:

It's the mode in which the program starts, which is used to -as its name indicates- see parts of the memory, although basic modifications can be made such as rotating, inverting, mirroring and more.

In this mode you can displace through the entire Spectrum's memory.

1-4. What is Edit Mode:

This mode is entered by pressing the 'E' key or the 'Enter' key from ***Viewer Mode*** , and allows you to modify the char from the grid directly, just like a sprite editor. You'll know you're in Edit Mode because the border of the screen will change to Cyan. Needless to say this mode is the sauce of this program, otherwise it would be called Character Explorer... wait, oops!

In this mode you can make use of the ***Sprite bar*** (CShift + S), which displays 2x2 char groupings for facilitate making 16x16 pixel sprites, and the ***Seamless Background mode*** (CShift + B), which shows you a 9x9 grid with the character you are editing, to make it easier for you to make backgrounds that repeat indefinitely.

In this mode you CAN NOT displace through all the memory, since there are parts of it protected so that the system does not crash by mistake. Note that if you are in ***Viewer Mode*** while in a protected part of memory, you will not be able to enter ***Edit Mode*** until another non-protected part of memory is selected.

1-5. Area bar:

Protected parts appear with a padlock at the bottom of the ***Area bar*** (the coloured bar that appears at the top). See ***images 1*** and ***2*** for further clarification.

Also the colour of the ***Area Bar*** indicates whether that part of the memory is protected or not:

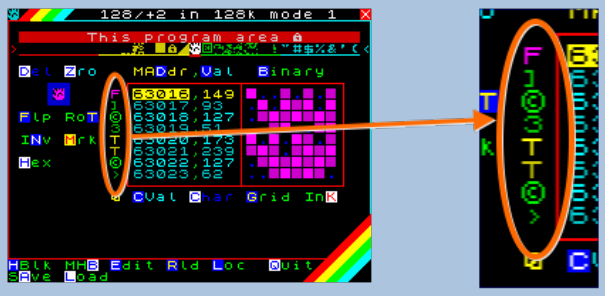
Image 3 – *Area bar.*

- >**Dark Blue**: This part belongs to the ROM, and can **NOT** be modified.
- >**Red**: This area belongs to BASIC, the program itself and other things, and can **NOT** be modified.
- >**Magenta**: This area belongs to the UDG (User Defined Graphics) and **YES** it can be modified.
- >**Green**: This area is not protected and **CAN** be modified. This is the part you should use.
- >**Cyan**: This area belongs to the CharSet of the ROM and, obviously, it can **NOT** be modified.
- >**Yellow**: It belongs to the *System Variables*, printer buffer *(in 48k)*, etc. and **YES** it can be changed, but I'm still thinking that maybe it's not such a good idea to allow it...
- >**White**: Belongs to the video memory of the screen, and can **NOT** be modified.

In short, only the *areas* that are in ***Green*** , ***Magenta*** *(my eyes!) and* ***Yellow*** are allowed to modify.

These restrictions also apply to the modifications that can be made in the ***Viewer Mode*** (invert, rotate, etc.). Or at least they should, any bug you know, you will make me a very happy person if you let me know. ;)

1-6. Legend of what is shown in the *Info. about the Byte:*



>This section of the interface provides you with information about the byte value of each of the eight memory addresses in the grid.

It tells you -if it's an *ASCII code*- el caracter que representa, what character it represents, or if it's a *token*, or if it's a special *control code* used by the *Spectrum* to control certain things.

Image 4 – *Info. about the Byte.*

The colour of the char that appears in the ***Info. about the Byte*** section (to the left of the memory addresses in the *grid*), gives you information about the byte value of that memory address. It tells you what *char* represents, if it's a *token*, an UDG, etc.

>**Dark Blue**: A **dot** appears. The byte value is less than 32 and doesn't represent any *ASCII* char. They are *control codes* used by the *Spectrum* (*cursors*, *Enter*, *Delete*, colours, etc).

>**Green**: The byte value is between 32 and 143, and tells you what *ASCII* char it represents.

>**Magenta**: The byte value is between 144 to 162 in *128k*, and up to 164 in *48k*, and represents the *UDG*. The graphic you have defined doesn't appear, only the letter that it represents.

>**Yellow**: A '**T**' appears. The byte value is between 162 in *128k* or 164 in *48k* up to 255, saying that it represents a Spectrum *token* (a *BASIC* command, in a nutshell).

1-7. *UDG* Bar:

>This character bar appears when you are in the part of memory where the UDG's reside *(beginning with 65 368).*

Image 5 – *UDG bar.*

When you enter the part of memory that contains the *User Defined Graphics (UDG)*, this bar appears so that you can have all the *UDG's* at a glance.

2. The key list:

2-1. In View Mode:

-*Grid displacement:*

>**O**:	–1 Char (–8 bytes).	>**P**:	+1 Char (+8 bytes).
>**CShift + O**:	–4 Chars (–32 bytes).	>**CShift + P**:	+4 Chars (+32 bytes).
>**Q**:	–4 Bytes.	>**A**:	+4 Bytes.
>**CShift + W**:	–1 Byte.	>**CShift + S**:	+1 Byte.
>**I**:	–8 Chars (–64 bytes).	>**K**:	+8 Chars (+64 bytes).
>**CShift + I**:	–1 CharSet (–96 chars, –768 bytes.)	>**CShift + K**:	+1 CharSet (+96 chars, +768 bytes).

>**W**: Move up the ***yellow selection bar***. >**S**: Move down the ***yellow selection bar***.

-*Overall (View Mode):*

>**B**:	*Binary mode.*
>**H**:	Change from *decimal* to *hexadecimal* and vice versa.
>**E**, **Enter**:	Enters to ***Edit Mode*** with cursor in column 4.
>**R**:	Refreshes the *grid* and the *position* and *UDG bars*.
>**CShift + R**:	Reload the program.
>**L**:	Enter to the '***Useful Locations***' menu.
>**CShift + Q**:	Exit to **BASIC** (type ***RUN*** to return to the program).
>**CShift + H**:	Change the border colour to black (in case you didn't like blue).
>**CShift + B**:	***MicroHobby*** *CharSets* presentation and selection screen.
>**D**:	Change to memory address (*Input*).
>**G**:	Change the *grid* background colour.
>**CShift + G**:	Changes the colour of the *background* dots (inactive pixels) of the *grid*.
>**SShift + K**:	Changes the *ink* colour (active pixels) of the *grid*.
>**CShift + A**:	Enters to the ***Save*** menu.
>**CShift + L**:	Enters to the ***Load*** menu.

>**M**: Memory *start* address *selection mark* for *copying*, *moving*, *deleting* or *saving (Save/Load).*

>**CShift + M**: *End* of memory address *selection mark* for *copying*, *moving*, *deleting* or *saving (Save).*

>**SShift + M** *(the dot)*, clear the *selection/save marks* you put.

-Overall (other keys):

>**SShift + Z**: Deletes the data between the *selection/save marks*.
>**SShift + X**: Moves the data from the *selection/save marks* to the current address.
>**SShift + V**: Copies the data from the *selection/save marks* to the current address.

>**Numbers** from **1** to **8**: To enter the ***Edit Mode*** with the cursor in the column of the pressed number.

-Some editing:

>**V**: Change the value of the memory address selected in *yellow* (*Input*).
>**CShift + D**: Deletes the current char.
>**F**: Flip the char vertically.
>**T**: Rotate clockwise.
>**N**: Invert char.
>**CShift + Z**: Deletes the row (*byte*) selected.
>**CShift + F**: Flip the char horizontally.
>**CShift + T**: Rotate anticlockwise.

>**SShift + 6**, **7**, **8** y **9** *(Sinclair Joystick)*: 'Accommodate' the char (displace around the grid).
>**N**, **SShift + 0** *(Shot on Sinclair Joystick)*: Inverts the char (a negative, **0** is set to **1** and vice versa).

-Clipboard:

>**C**: Copy the value (byte) from the selection.
>**CShift + C**: Copies the current char (8 bytes).
>**CShift + V**: Pastes the value (byte) or char (8 bytes) at current position.
>**CShift + X**: Clears the ***clipboard***.

2-2. In Edit Mode:

-Displacement through the grid:

>**W**, **A**, **S**, **D**: Moves the cursor (*Up*, *Left*, *Down* and *Right*).
>**O**, **P**: Moves the cursor (*Left*, *Right*).
>**Q**, **E**, **Z**, **C**: Moves the cursor diagonally (*Up Left, Up Right*, *Down Left* and *Down Right*).

>**CShift + O**, **P**: Save the char in memory and move to previous/next char.

-Pixel writing:

>**X**: *Activates* **writing**. It's disabled by pressing again the '**X**' key or the **Space Bar**.
>**CShift + 0**, **Delete**: *Activates* **eraser**. It's disabled by pressing '**0**','**CShift + 0**' keys or the **Space Bar**.
>**N**: *Turns ON **pixel*** at the cursor position (even with **writing/eraser** active).
>**M**: *Turns OFF **pixel*** at the cursor position (even with **writing/eraser** active).
>**Barra Espaciadora**: *Enables/disables **pixel*** at the cursor position.

-Modifications:

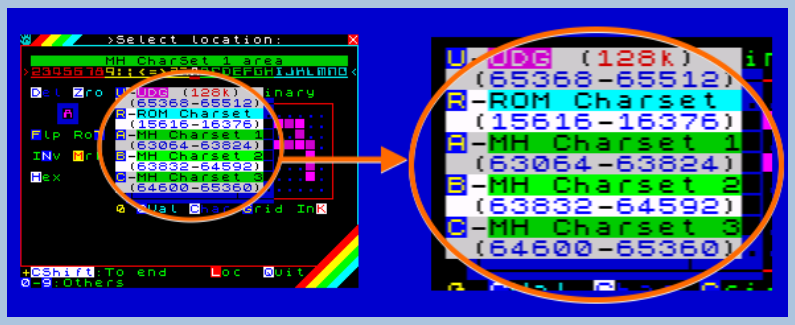
>**CShift + D**: Deletes current char.
>**CShift + Z**: Deletes the row (byte) selected in yellow.
>**F**: Flip the char vertically. >**CShift + F**: Flip the char horizontally.
>**T**: Rotate clockwise. >**CShift + T**: Rotate anticlockwise.

>**SShift + 6**, **7**, **8** y **9** *(Sinclair Joystick)*: 'Accommodate' the char (displace around the grid).
>**I**, **SShift + 0** *(Shot in Sinclair Joystick)*: Inverts the char (a negative, **0** is set to **1** and vice versa).

-Overall (Edit Mode):

>**B**: *Binary mode.*
>**CShift + S**: Activate/deactivate ***Sprites mode***.
>**1**, **2** y **3**: *Offset* del ***modo Sprites*** (–1, +1 y reset a 0).
>**CShift + B**: Enable/disable ***Seamless mode***, to make it easier to draw *seamless* tiled *backgrounds*, for example.
>**CShift + E**: It does ***NOT*** save the char in memory, restores the original and returns to ***View Mode***.
>**Enter**: *Save* the char in memory and returns to ***View Mode***.
>**CShift + Q**: *Save* the modified char in memory and exit to ***BASIC*** (type *RUN* to return to the program).
>**R**: Updates the *UDG*, *Sprites* and *Position bars*.
>**CShift + R**: Reload the char and the rest (the *grid* and the *UDG*, *Sprites* and *Position bars*).

2-3. In the 'Useful Locations' selection menu:



>In this menu you can quickly jump to predefined memory addresses.

Image 6 *– Useful Locations menu.*
*('**L**' key in View Mode)*

>**U**: *UDG* (65 368).
>**R**: *ROM* CharSet (15 616).
>**A**: *MicroHobby* CharSet 1 (63 064).
>**B**: *MicroHobby* CharSet 2 (63 832).
>**C**: *MicroHobby* CharSet 3 (64 600).
>**CShift + U**: End of *UDG* (65 512 -at *128k*-, 65 528 -at *48k*-).
>**CShift + R**: End of *ROM* CharSet (16 376).
>**CShift + A**: End of *MicroHobby* CharSet 1 (63 824).
>**CShift + B**: End of *MicroHobby* CharSet 2 (64 592).
>**CShift + C**: End of *MicroHobby* CharSet 3 (65 360).
>**CShift + Q**: Exits to ***BASIC*** (type *RUN* to return to program).

– Key list. Edit Mode, Useful Locations menu –

-More 'Useful Locations' keys:

>**1**: *First third* of the screen (16 384).
>**2**: *Second third* of the screen (18 432).
>**3**: *Third third* of the screen (20 480).
>**4**: Screen *colour attributes* (22 528).
>**5**: *System variables* at *128k*, printer *buffer* at *48k* (23 296). *System variables* on *48k* begins at 23 552.
>**6**: This program code start (25 200).
>**7**: **Free User Memory (56 000).**
>**8**: Program routines, data and variables (62 968).
>**9**: *MicroHobby CharSets* routine (63 040).
>**0**: Strat of *ROM* (00 000).

2-4. In the MicroHobby CharSets selection screen:

>In this screen, you can change the program's text font to one of the three fonts in the *MicroHobby CharSets* memory space. So you can try them to see how they turn out.

If you have charsets somewhere else in memory and you want to test them, copy the charset to one of these three addresses. These are the ones that correspond to the ***A***, ***B*** *and* ***C*** keys in the *'Useful Locations'* menu (***CShift*** + ***L***).

*- Go to **page 12** for more information about MicroHobby Charsets, and the part of memory where they are stored -*

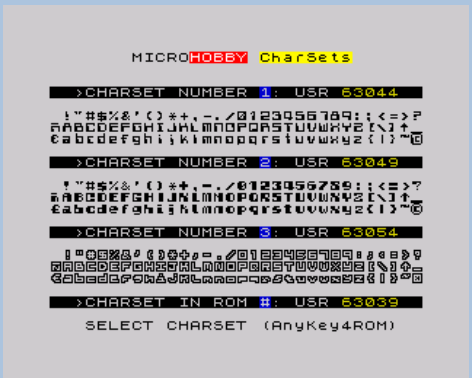


Image 7 *– MicroHobby CharSets.*
*('**CShift** + **B**' keys in View Mode)*

>**1**: Selects *Charset 1* and change the program font (to test a new CharSet, for example).
>**2**: Selects *Charset 2* and the same as above.
>**3**: Selects *Charset 3* and...

>**Any other key** changes to the default font (*ROM*) and returns to ***View Mode***.

– Key List. Useful Locations menu, MicroHobby CharSets –

2-5. In the Save and Load menus:

Image 8 – *Save/Load menu.*

>Here you can save/load your *Charsets*, *UDG's*, *Sprites*, etc.

You can save to *tape* on all *Spectrum* models, plus to *RAM* on the *128k* models, and also to *disk* on the ***+3***. You can also save manually, exiting to *BASIC* with **CShift + Q**.

I recommend you exit to *BASIC* first and set the *SCREEN mode* on a *128k* model, so that the upper screen is not deleted when saving/loading manually, and to have the data in view.

>**1**: Save/Load *UDG's*.
>**2**: Save/Load ***MicroHobby*** *CharSets* (all three, 288 chars --2304 bytes without routine, --2329 bytes with routine).
>**R**: Select save/load with or without routine.

>**3**: Save/Load ***MicroHobby*** *CharSet* (only one charset, 96 chars --768 bytes).
>**M**: Select area to save/load *CharSet* (or whith the **A**, **B** or **C** keys).

>**4**: Save/Load from/to selection/save mark(s).
>**5**: Save/Load from/to custom address. It must be *56000 ($DAC0) or more.*
>**6**: Save complete program / Load from file header.

>**T**: *Save/Load to Tape, Disk or RAM.*

>**Any other key** returns to ***View Mode***.

Be careful when loading, because you can load data on top of the program and make it crash. If saving/loading gives you an error or you **break**, return to the program again with a simple *RUN*.

If you need to save or load from a different drive (if you have a ***divIDE***, or ***Spectrum Next's C*** *drive*) you must to do manually by exiting to *BASIC* (**CShift + Q**).

If you plan to load charsets from the ***ZX Origins*** page *(https://damieng.com/typography/zx-origins)* I recommend that you load them into one of the three *MicroHobby* charsets areas (*option/key* ***3***).

3. Manually save and load data, character sets, or UDG's:

To record or load data manually you have to exit to *BASIC* with the **CShift + Q** keys (better you write down the memory address and the length of the data to be recorded/loaded on a piece of paper, or you do the thing of going out to *BASIC* and select *SCREEN mode* on a *128k* model).

To load data, you don't do any *CLEAR xxxxx*. If you do, you'll probably mess up the program code and you won't be able to return to it without crashing. There's already a *CLEAR 25 199* in the loader, which is where the program begins. From where the program ends (around 56 000) to 65 535, is free memory that can be used (except for an intermediate part that the program uses for a few data).

Look carefully where you load things so you don't step the program code. Remember that later in your program/game, you can load this data in a memory location that is more comfortable for you without any problem.

You can use this same program to see which parts are free and thus load data without fear of crushing anything. In ***View Mode***, use **CShift + I** / **K** to move quickly with 768-byte jumps, and without pushing **CShift** for 64-byte jumps.

4. Differences between *48k* and *128k* models:

The program tries to detect the *Spectrum* model, and what mode it is currently in (*mode 128 or 48*). The program also detects if you are on a ***+3*** and will use the disk drive. In the *128k* models it's possible to use the *RAM disk*. If it fails to detect the model, it will treat it as a *Spectrum 128k* (the Russian clones, for example).

Depending on what it detects, it goes into a mode of four different ones *(you can use the tape in all modes)*:

- **Mode 0**: You are on a ***48k*** model, or a ***128k*** model in *48 mode (USR 0,* only the *Tape* can be used*)*.
- **Mode 1**: You are on a ***128/+2*** in *128 mode*, o en un clon no detectado (*RAM disk* can be used).
- **Mode 2**: You are on a ***+2A/B*** in *128 mode* (*RAM disk* can be used).
- **Mode 3**: You are on a ***+3*** or a ***+2A/B*** with a *Disk drive* attached (either the *RAM disk* or the *Disk drive* can be used).

How do these modes differ? Well, practically only in the layout of the storage device, but certain specific things also change, which I explain below:

In the '***Useful Locations***' menu, when pressing the **CShift + U** key to go to the end of the *UDGs*, in *128k* it goes to *UDG* **S**, and in *48k* mode it goes to *UDG* **U** (although you can edit the **T** and **U** both in *48k* and *128k*). In the ***Area Bar***, on *48k* the printer buffer appears, in *128k* it doesn't.

5. About MicroHobby CharSets:

This program comes with three character sets. They are the ones that came in *MicroHobby* magazine number 30 (pages 14, 15 and 16), in a section called '*Customize your Spectrum*', but the third one is changed by a set made by me, and the other two with very slight changes in some char. You can use them, modify as you like or create new ones.

Apart from the three sets, in the magazine came a little assembly routine to change to any of the three sets with a *RANDOMIZE USR*, which changes the *CHARS* System variable. This routine its included.

In the program, in ***View Mode***, you can press **CShift + B** to see all three sets, even select one that will change the program's text font to see what it would look like. In this charset selection menu you can see next to each set a text that says '*USR xxxxx*', where '*xxxxx*' is the memory address where you have to do the *RANDOMIZE USR* to change to that charset. I'll put it here as well:

>**RANDOMIZE USR 63044** for first set.
>**RANDOMIZE USR 63049** for second set.
>**RANDOMIZE USR 63054** for third set.
>**RANDOMIZE USR 63039** to return to the *ROM* character set.

The charset selection routine begins at memory address 63039, and ends at 63063 (inclusive), and is only 25 bytes long. There is a program attached that creates the routine for up to 32 charsets.

If you use these character sets, I recommend that you call the *ROM* set routine when exiting the program that uses them, especially in *128k BASIC mode*, or you will see garbage (or straight blanks) instead of the program listing. At runtime, the charsets will look fine, in *48k mode* this does not occur. It's all ok, neither the listing of your program has been corrupted or anything, it's just the way that the *128k Spectrum* works.

The first charset goes from address 63064 (a complete charset is *768 bytes*) to 63831, the second from 63832 to 64599, and the third from 64600 to 65367 (right up to where the *UDGs* begin, which is *65368*). Keep this in mind if you only want to save the charsets (or just some) without the assembler routine mentioned above.

For more information on MicroHobby charsets, visit the following link (in spanish):

https://microhobby.speccy.cz/mhforever/numero030.htm

Look for the '*Personaliza tu Spectrum*' section (*pages 14, 15 and 16*). You even have the source code of the assembler routine, which is very simple, and you can modify it to add more charsets, or to be able to place the routine in another memory address if necessary. If so, remember that the *CHARS System variable* points to the memory address of the character set *MINUS 256*. That is, you have to tell the *CHARS* variable that you have the charset **256 bytes less** than the position where the charset actually is in memory.

6. Final Notes and Acknowledgments:

On the ***ZX Origins*** website *(https://damieng.com/typography/zx-origins)* you have hundreds of complete charsets at your disposal for download. This website is pure gold. Thanks a lot to ***DamienG*** for creating it.

If anyone knows any POKE or RANDOMIZE USR or something to switch to the *Screen mode* from *BASIC 128k* (*the lower screen*), please tell me how. It would be nice if it appears in this mode when you exit the program, so that the upper screen is not deleted and you can have, for example, the addresses to be recorded in view.

Thank you very much to ***vicentecno*** for his asm routine to find *multiples of 32*, it has been very useful for the *Sprite bar*. Many thanks to ***Sergio thEpOpE*** for the *Seamless background* suggestion and for his videos (they help me a lot). Many thanks also to ***Rodolfo Guerra*** for discovering the ***HiSoft*** compiler to me.

And thank you very much especially to ***vicentecno***, ***Paco Vespa***, ***Azimov***, and in general to all the people on the ***Arnau Jess*** discord for putting up with me (***Isaías***, ***Jimmy***), and especially to the ***#desarrollo*** channel of the same server. Thank you so much everyone...

*- Made in **Sinclair BASIC** and compiled with **HiSoft BASIC Compiler** (128k version) -*

Fin

*·**PS:** Okay, so many keys may have gotten a bit out of hand. I am so sorry...*