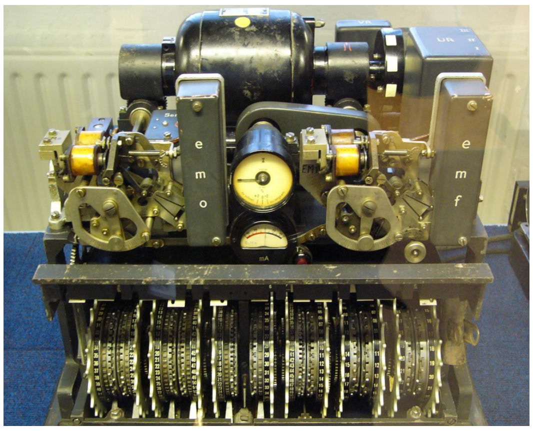
An SZ42 (Tunny) simulator (including a Colossus simulation function)

JMcC, UK – January 2018.

Document purpose.

This document describes some software which can be used to (approximately) simulate the function of the Lorenz SZ42 encryption machine. It does not claim to be a complete and total reproduction, but it is nevertheless interesting and educational to run.


Source: Wikimedia Commons
Retrieved February 2014

This software has been extended to incorporate an automatic looping decryption function, allowing it to simulate some of the functionality of Colossus.

Additional software is supplied which allows the SZ42-mode of the program to operate in auto-typing mode from a script.

Bibliography

[a] **http://en.wikipedia.org/wiki/Lorenz_SZ42**
[b] **http://de.wikipedia.org/wiki/Lorenz-Schl%C3%BCsselmaschine**
[c] **http://en.wikipedia.org/wiki/Colossus_computer**
[d] **http://de.wikipedia.org/wiki/Colossus**

Annexes

[1] **JFish Vn5.31 (or later).**
[2] **JFishStepper Vn4 (or later) .**
[3] **SZ42-EN.pdf** ***(being Bibliography [a] retrieved on Feb 14th, 2014)***
[4] **SZ42-DE.pdf** ***(being Bibliography [a] retrieved on Feb 14th, 2014)***
[5] **Colossus.EN.pdf** ***(being Bibliography [c] retrieved on Feb 18th, 2014)***
[6] **Colossus.DE.pdf** ***(being Bibliography [d] retrieved on Feb 18th, 2014)***

PART I – SZ42.

Introduction.

From [3]:

The Lorenz SZ40, SZ42A and SZ42B were German rotor stream cipher machines used by the German Army during World War II. They were developed by C. Lorenz AG in Berlin and the model name SZ was derived from Schlüsselzusatz, meaning cipher attachment. The instrument implemented a Vernam stream cipher.

From [4]:

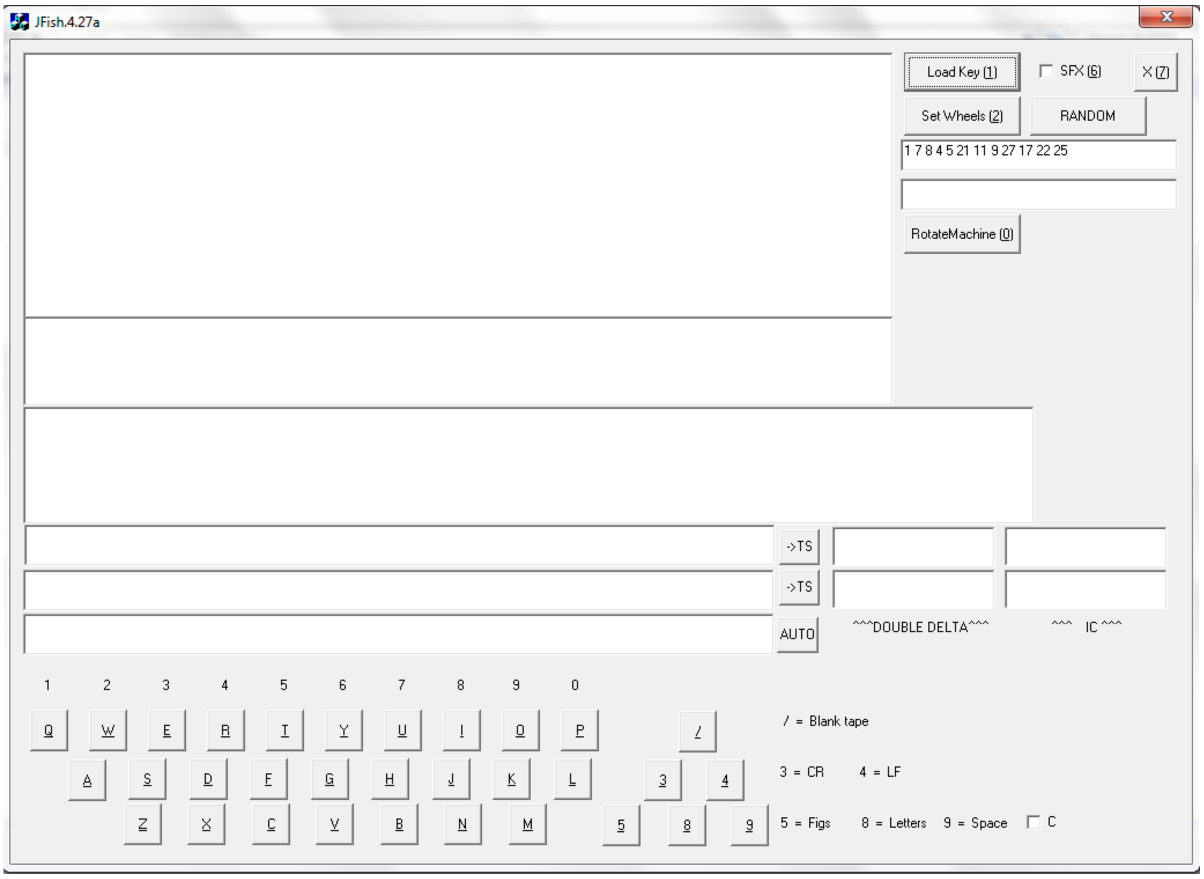
Die Lorenz-Schlüsselmaschine (auch: Lorenz-Schlüsselzusatz), von den britischen Codeknackern in Bletchley Park „Tunny“ (deutsch: „Thunfisch“) genannt, diente zur geheimen Kommunikation mittels Fernschreibverbindungen. Sie wurde von der C. Lorenz AG, Berlin im Auftrag der deutschen Militärführung als Ergänzung zur Enigma-Schlüsselmaschine entwickelt, die mittels Morsecode verschlüsselte Nachrichten über Funkverbindungen übermittelte.

<u>The Software.</u>

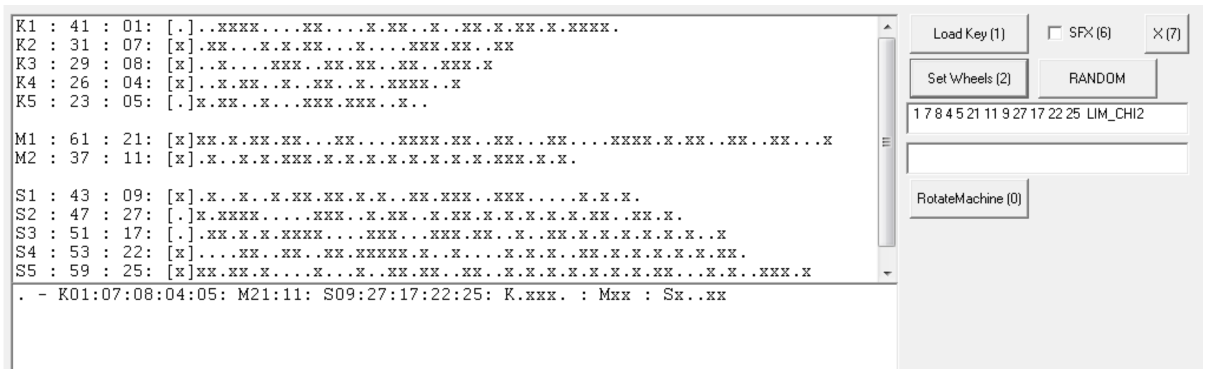
The program is issued, with its source included, as a C++6 program, compiled as a 32-bit executable.

To run the program, unzip the supplied ZIP file, navigate to the ***"Release"*** folder, and copy the files ***JFishKey.txt, JFish.exe, HiFile.wav and LowFile.wav*** to some arbitrary folder of your choice. Then, run the file **JFish.exe** by clicking on it.

On so doing, the following screen will be shown.


1 7 8 4 5 21 11 9 27 17 22 25

Clicking, in order, the **[Load Key]** and the **[Set Wheels]** buttons will result in some of the blank areas shown above being populated with data from the JFishKey.txt file. For further details concerning the content of the JFishKey.txt file, please see Appendix A below.



```
K1 : 41 : 01: [.]..xxxx....xx....x.xx..x..xx.x.xx.x.xxxx.
K2 : 31 : 07: [x].xx...x.x.xx...x....xxx.xx..xx
K3 : 29 : 08: [x]..x....xxx..xx.xx..xx..xxx.x
K4 : 26 : 04: [x]..x.xx..x..xx..x..xxxx..x
K5 : 23 : 05: [.]x.xx..x...xxx.xxx..x..

M1 : 61 : 21: [x]xx.x.xx.xx...xx....xxxx.xx..xx...xx....xxxx.x.xx..xx..xx...x
M2 : 37 : 11: [x].x..x.x.xxx.x.x.x.x.x.x.x.x.xxx.x.x.

S1 : 43 : 09: [x].x..x..x.xx.xx.x.x..xx.xxx..xxx.....x.x.x.
S2 : 47 : 27: [.]x.xxxx.....xxx..x.xx..x.xx.x.x.x.x.x.xx..xx.x.
S3 : 51 : 17: [.].xx.x.x.xxxx....xxx...xxx.xx..x..xx.x.x.x.x.x.x..x
S4 : 53 : 22: [x]....xx..xx..xx.xxxxx.x..x....x.x.x..xx.x.x.x.x.x.xx.
S5 : 59 : 25: [x]xx.xx.x....x...x..xx.xx..xx..x.x.x.x.x.x.x.xx...x.x..xxx.x
```

```
. - K01:07:08:04:05: M21:11: S09:27:17:22:25: K.xxx. : Mxx : Sx..xx
```

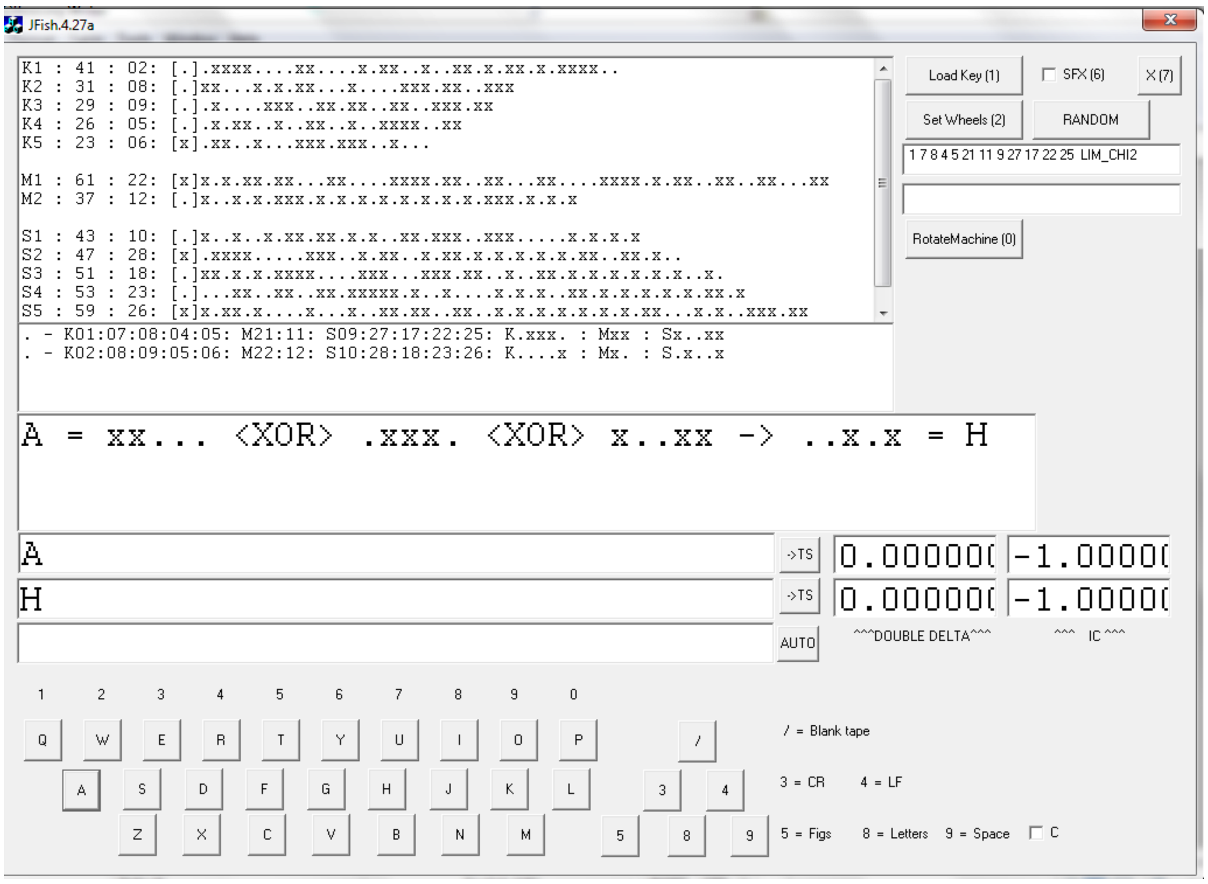
```
1 7 8 4 5 21 11 9 27 17 22 25 LIM_CHI2
```

Now, to explain the data which has appeared; the top-most box on the left side shows a list of all of the wheels, with their names, sizes, and positions, followed by their pin-settings with '*' as a 1, and '.' as a 0. The pin-setting at the left end of each list, shown within the square brackets [ ], shows the *current* pin-settings, that is to say, the pin-settings that will apply to encrypt/decrypt the *next* character.

The second box shows the same information in a somewhat abbreviated form.

To the right, the initial wheel positions are shown. *Note that this simulation does not support any so-called limitations; although LIM_CHI2 is shown in the above image, the simulation behaves as though LIM_NONE is in use.*

The simulation is now ready to encrypt a character: on pressing a character, for example, the **[A]** key on the keyboard (a feature that the SZ42 did *not* have), some interesting things can be seen:



```
K1 : 41 : 02: [.].xxxx....xx....x.xx..x..xx.x.xx.x.xxxx..
K2 : 31 : 08: [.]xx...x.x.xx...x....xxx.xx..xxx
K3 : 29 : 09: [.].x....xxx..xx.xx..xx..xxx.xx
K4 : 26 : 05: [.].x.xx..x..xx..x..xxxx..xx
K5 : 23 : 06: [x].xx..x...xxx.xxx..x...

M1 : 61 : 22: [x]x.x.xx.xx...xx....xxxx.xx..xx...xx....xxxx.x.xx..xx..xx...xx
M2 : 37 : 12: [.]x..x.x.xxx.x.x.x.x.x.x.x.x.xxx.x.x.x

S1 : 43 : 10: [.]x..x..x.xx.xx.x.x..xx.xxx..xxx.....x.x.x.x
S2 : 47 : 28: [x].xxxx.....xxx..x.xx..x.xx.x.x.x.x.x.xx..xx.x..
S3 : 51 : 18: [.]xx.x.x.xxxx....xxx...xxx.xx..x..xx.x.x.x.x.x.x..x.
S4 : 53 : 23: [.]...xx..xx..xx.xxxxx.x..x....x.x.x..xx.x.x.x.x.x.xx.x
S5 : 59 : 26: [x]x.xx.x....x...x..xx.xx..xx..x.x.x.x.x.x.x.xx...x.x..xxx.xx
```

```
. - K01:07:08:04:05: M21:11: S09:27:17:22:25: K.xxx. : Mxx : Sx..xx
. - K02:08:09:05:06: M22:12: S10:28:18:23:26: K....x : Mx. : S.x..x
```

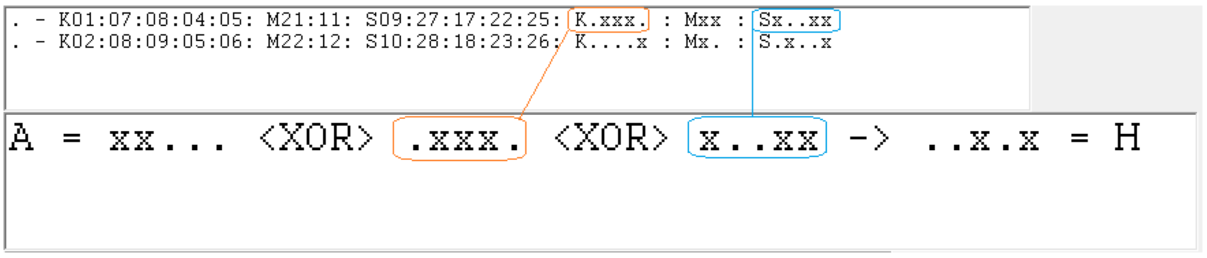
```
A = xx... <XOR> .xxx. <XOR> x..xx -> ..x.x = H
```

```
1 7 8 4 5 21 11 9 27 17 22 25 LIM_CHI2
```

```
A                                   0.00000(   -1.0000(
H                                   0.00000(   -1.0000(
```

1) The wheels in the top-most box can be seen to have moved, showing the settings for the *next* encryption.
2) An second line has been added to the second box, showing the abbreviated settings for the encryption which has just been done, and the abbreviated settings for the *next* encryption.
Note that up to four lines can be shown in this box. The bottom line represents the next encryption, the second from bottom, the last encryption, and the third and fourth lines from the bottom, the two encryptions before the last encryption.
3) The third box is now populated, showing the work done in the encryption which has just been done.
4) The fourth and fifth boxes have now been populated, showing the plain text 'A', and its encrypted value 'H' in the two lines.

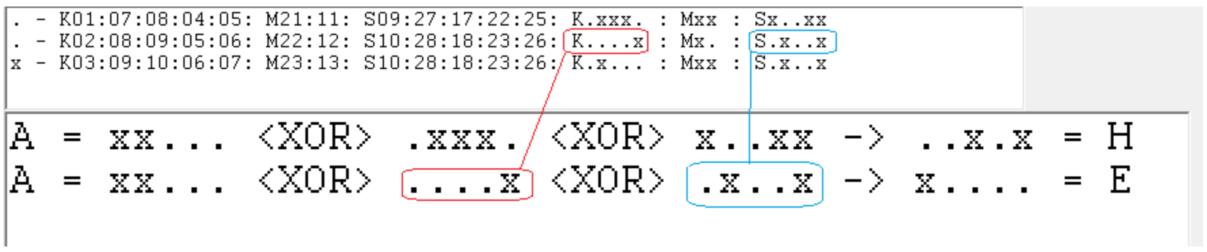
To study the second and third boxes in slightly more detail, the sequence of taking the bit-wise representation of the letter 'A' (xx...), XOR'ing it with the 'K' wheel, and then with the 'S' wheel, to produce the encryption, 'H' (..x.x), can be seen:



```
. - K01:07:08:04:05: M21:11: S09:27:17:22:25: K.xxx. : Mxx : Sx..xx
. - K02:08:09:05:06: M22:12: S10:28:18:23:26: K....x : Mx. : S.x..x
```

```
A = xx... <XOR> .xxx. <XOR> x..xx -> ..x.x = H
```

If the **[A]** key is pressed for the second time, the 'A' is this time encrypted to 'E', with the following tracing visible:



```
. - K01:07:08:04:05: M21:11: S09:27:17:22:25: K.xxx. : Mxx : Sx..xx
. - K02:08:09:05:06: M22:12: S10:28:18:23:26: K....x : Mx. : S.x..x
x - K03:09:10:06:07: M23:13: S10:28:18:23:26: K.x... : Mxx : S.x..x
```

```
A = xx... <XOR> .xxx. <XOR> x..xx -> ..x.x = H
A = xx... <XOR> ....x <XOR> .x..x -> x.... = E
```

The effect of typing in a more complex message will now be studied.

Start the program anew, and then click the **[Load Key]** and the **[Set Wheels]** buttons.

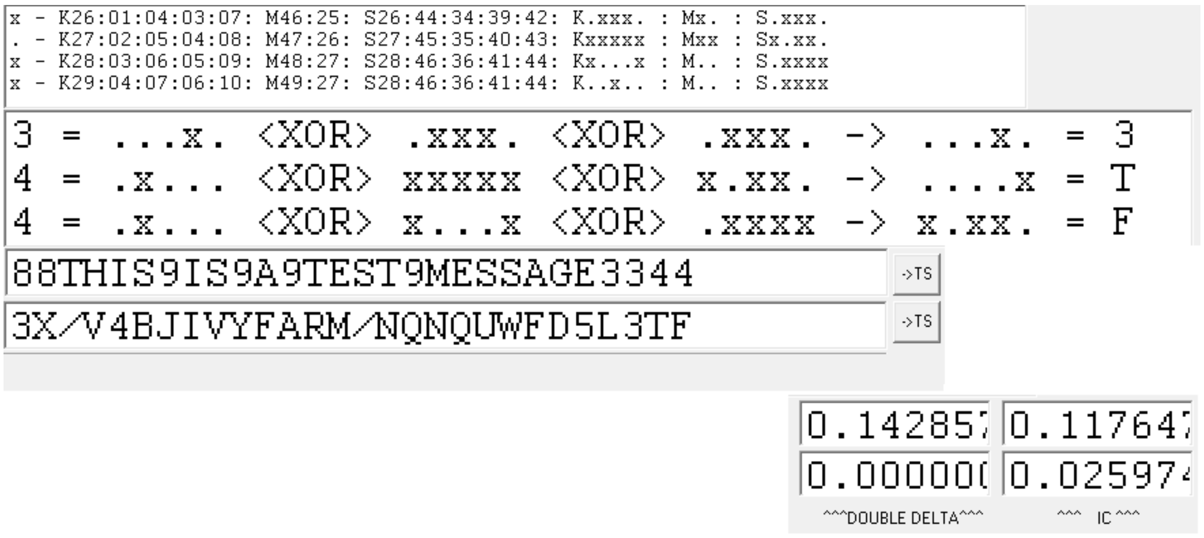
The input message takes the form:
`<letter shift><letter shift>THIS<space>IS<space>A<space>TEST<space>MESSAGE<cr><cr><lf><lf>`

Using a convention from Bletchley Park, the actual characters used, to be typed on the keyboard, would be:

`88THIS9IS9A9TEST9MESSAGE3344`

Note that the majority of the buttons used by this application have **hot-keys** and can be input without mouse clicks; **[Load Key]** can be operated using **ALT + 1**, and **[Set Wheels]** by **ALT + 2**. The letters **{A...Z},** the numbers **{3, 4, 5, 8, 9}** and **'/'** can be operated by **ALT + <the letter, number, or '/'>**. So, for example, **88THIS** ***etc*** can be input with **ALT+8 ALT+8 ALT+T ALT+H ALT+I ALT+S** ***etc***. More details are to be found in Appendix D below.

The plain text message, along with its encrypted version can be seen below:



```
x - K26:01:04:03:07: M46:25: S26:44:34:39:42: K.xxx. : Mx. : S.xxx.
. - K27:02:05:04:08: M47:26: S27:45:35:40:43: Kxxxxx : Mxx : Sx.xx.
x - K28:03:06:05:09: M48:27: S28:46:36:41:44: Kx...x : M.. : S.xxxx
x - K29:04:07:06:10: M49:27: S28:46:36:41:44: K..x.. : M.. : S.xxxx
```

```
3 = ...x. <XOR> .xxx. <XOR> .xxx. -> ...x. = 3
4 = .x... <XOR> xxxxx <XOR> x.xx. -> ....x = T
4 = .x... <XOR> x...x <XOR> .xxxx -> x.xx. = F
```

```
88THIS9IS9A9TEST9MESSAGE3344
3X/V4BJIVYFARM/NQNQUWFD5L3TF
```

```
0.142857  0.117647
0.000000  0.025974
```

In this picture, the boxes containing numbers, which, in real-life, appear to the right of the input and output text boxes respectively, have been shunted to a lower position.

The two mathematical concepts behind these number boxes will now be explained, namely the **"Double Δ"** (*to be read/said as "**Double Delta**"),* and the **"Index of Coincidence"**.

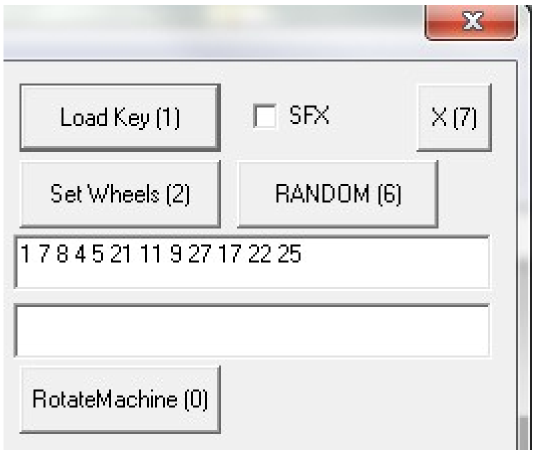
The **"Double Δ"** in this simulation is simply an indication of the number of times any given character is the same as the character immediately following it. This feature is used by Colossus during its activities as a tool to locate wheel starting positions as part of the SZ42 decryption process.

The **"Index of Coincidence" or "IC"** (*described in more detail in Appendix C*) is used here as a means of identifying text which is most likely to be plain-text. It is important to note that this process, although known during the life time of the SZ42/Colossus, is not known to have been used during decryption efforts. It has been added to this simulator for fun/educational purposes!

Taking the example above, the plain-text message has, compared to the crypto-text equivalent, a higher Double Δ and a higher IC.

<u>A note on wheel positions.</u>

An area to the top right of the screen controls the wheel pins and positions.



1 7 8 4 5 21 11 9 27 17 22 25

The **[Load Key]** button opens the file ***fishkey.txt*** and establishes the pin settings. The wheel positions are read from ***fishkey.txt*** and copied into the upper wheel positions text box. If the **[Random]** button is clicked, randomised wheel positions are written to the lower wheel positions text box.

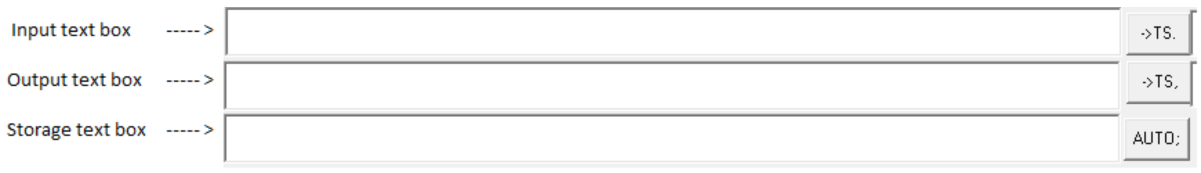
The **[Set Wheels]** button rotates the wheels to the wheel positions set in the upper wheel positions text box, *unless* there are wheel positions set in the lower wheel positions text box, in which case those are used instead.

Note that the **[Random]** button acts as a toggle: if there are no random positions specified in the lower wheel positions text box, then that text box is populated with randomised wheel positions; on the other hand, if there are already randomised wheel positions in the lower wheel positions text box, then they are cleared, meaning that the values in the upper wheel positions text box will be used instead. Operating the **[Random]** button auto-clicks the **[Set Wheels]** button.

Once the wheel pins and their positions have been established, data can be encrypted / decrypted. Note that the **[Rotate Machine]** button has the effect of moving on the wheels by one step, taking into account any pin settings which might be relevant (that is, those on the Motor Wheels).

A note on the input / output / storage text boxes.

There are three text boxes for data as it is input (upper most), output (middle) and stored (lowest).



Text which is typed in, using either the hot-keys, or by button clicking, appears in the upper, input text box; the middle, output text box will simultaneously be populated with text as it is processed by the simulated wheel mechanism.

Each of the two **[TS]** (Transfer to Storage) buttons will instantaneously transfer text in the text box to its left to the lowest (the Storage) text box.

The **[AUTO]** button to the right of the lowest (the Storage) text box will transfer the contents of that text box to the input text box and then execute a number of processing cycles (equal to the number of characters in the Storage text box, on that text, with the result of the processing appearing in the output text box.

This facility is useful for demonstrating the manual encryption of some text, and then the automatic decryption of the result, to ensure that the decryption is reversible.

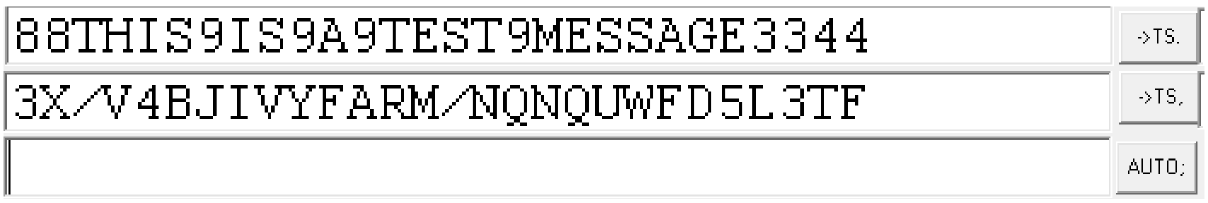
The following example is offered:

Start the simulator and click the **[Load Key]** and the **[Set Wheels]** buttons again.
If you like, press the **[Random]** key between the **[Load Key]** and the **[Set Wheels]**.

Using the keyboard or the hot-keys, enter the message:

88THIS9IS9A9TEST9MESSAGE3344

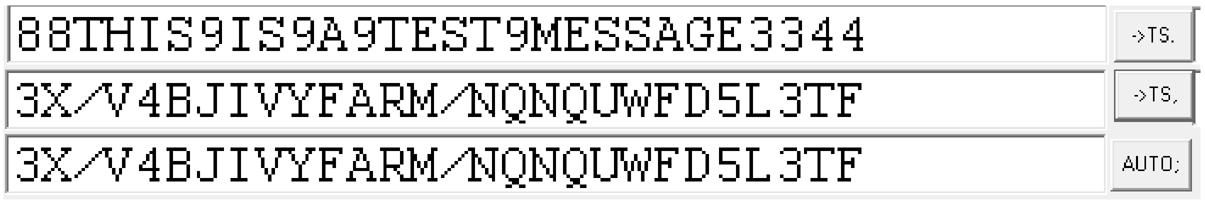
This message will appear in the input text box (the upper line), and an encrypted version will appear in the output text box (the middle line). Note that the encrypted version will vary depending on the wheel settings used. For the record, in this case, the values **“1 7 8 4 5 21 11 9 27 17 22 25”** were used.



```
88THIS9IS9A9TEST9MESSAGE3344
3X/V4BJIVYFARM/NQNQUWFD5L3TF

```

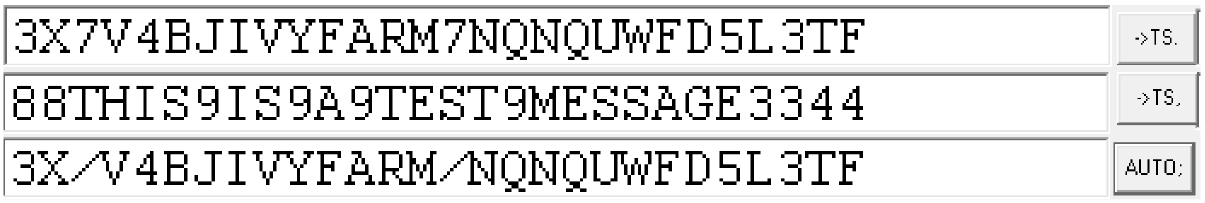
Press the **[TS]** button associated with the output text box, or use **ALT + ,** (*comma*) , to transfer the resultant encrypted text to the storage text box.



```
88THIS9IS9A9TEST9MESSAGE3344
3X/V4BJIVYFARM/NQNQUWFD5L3TF
3X/V4BJIVYFARM/NQNQUWFD5L3TF
```

Click the **[Load Key]** and the **[Set Wheels]** buttons again; however, this time, do not use the **[Random]** button as the Set Wheels function will use the randomised wheel settings already generated.

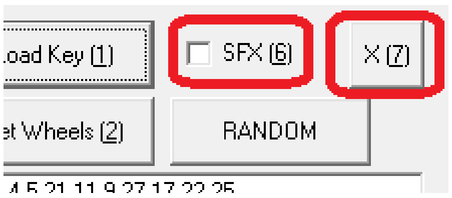
Click the **[Auto]** button, or use **ALT + ;** (*semi-colon*) – the input and output text boxes will be cleared, the contents of the storage text box will be transferred to the input text box, and the simulator will process the input text box to the output text box. All being well, and given that the mathematics of the SZ42 is fully reversible, the output text box should now show the original plain text (**`88THIS9IS9A9TEST9MESSAGE3344`**).



```
3X7V4BJIVYFARM7NQNQUWFD5L3TF
88THIS9IS9A9TEST9MESSAGE3344
3X/V4BJIVYFARM/NQNQUWFD5L3TF
```

Other SZ42 simulation functionality not yet mentioned.

There remain two functions on the top-right of the screen:



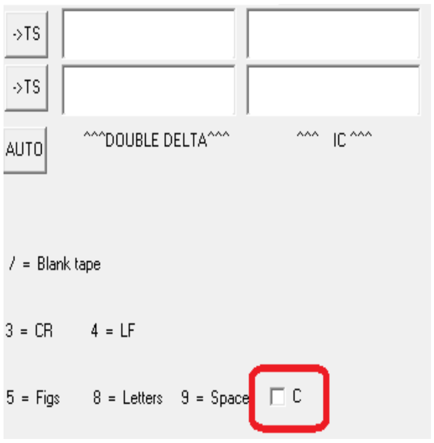
These are the **"SFX"** tick-box and **[X]** button, respectively.

The "**SFX**" tick-box turns on and off a sound-effect (a 'click' whenever a key is pressed on the simulated keyboard); it is recommended that this be turned **off** if running in any automated mode. It has no hot-key.

From version **5.31** of the software onwards, a second sound-effect tick box is available, labelled **"SF2"**. This plays high and low tones in sequence for each character that is output by the simulator. It has no hot-key.

The **[X]** closes the program. Its hot-key is **ALT + 7**.

At the bottom of the screen there is a tick-box marked **"C"**. Its function is to turn on and off some extra bottoms which allow the SZ42 simulator to operate in a pseudo-Colossus mode. This mode of operation is further described in Part II below. This tick-box has no hot-key.

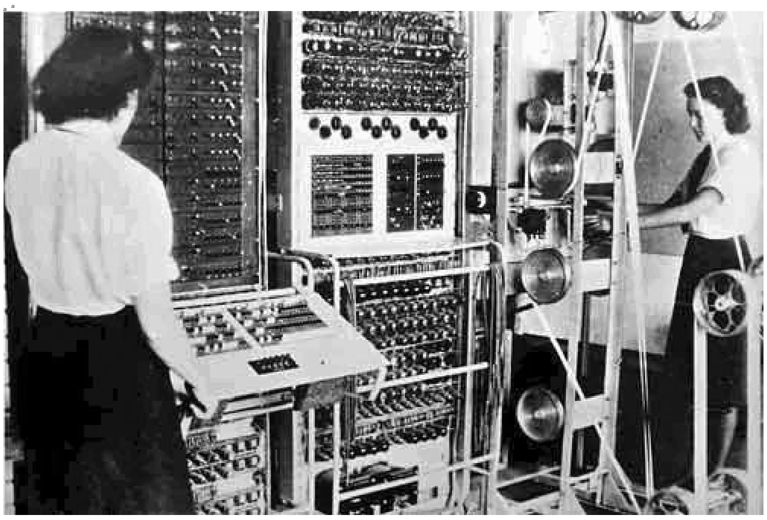


PART II – COLOSSUS.

Introduction.

From [5]:

Colossus was the world's first electronic digital computer that was at all programmable. The Colossus computers were developed for British code-breakers during World War II to help in the cryptanalysis of the Lorenz cipher. Without them, the Allies would have been deprived of the very valuable military intelligence that was obtained from reading the vast quantity of encrypted high-level telegraphic messages between the German High Command (OKW) and their army commands throughout occupied Europe. Colossus used thermionic valves (vacuum tubes) to perform Boolean operations and calculations.

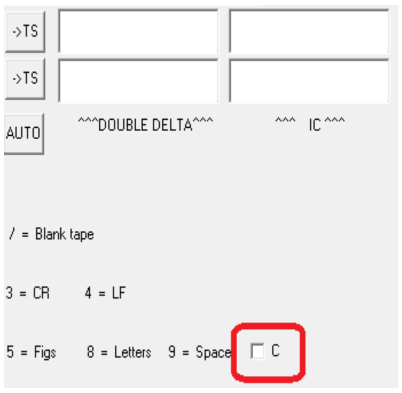

Source: Wikimedia Commons
Retrieved February 2014

From [6]:

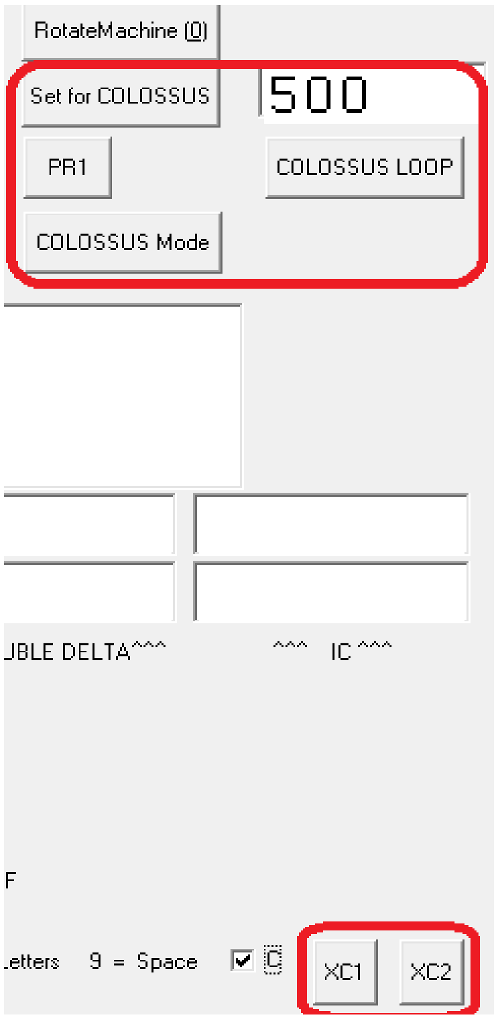
Colossi waren frühe Computer, die in England während des Zweiten Weltkriegs speziell zur Dechiffrierung von geheimen Nachrichten des deutschen Militärs gebaut wurden. Mit ihrer Hilfe wurde ab 1943 in Bletchley Park die Entzifferung der deutschen Lorenz-Schlüsselmaschine möglich.

The software.

Start the software as documented in Part I, and set the "C" tick-box, which it to be found at the bottom right of the screen.



Having so done, a number of new buttons and a text box appear:



None of these buttons have any hot-keys associated with them.

The following describes their functionality:

To the right of the **[Set for Colossus]** button is a text box, preset to the value **"500"**. Pressing the **[Set for Colossus]** button .causes the simulator to generate 500 sequential wheel positions, starting at the wheel position already set with the **[Load Key]**, **[Random]** (*optionally*) and **[Set Wheels]** buttons. Once this has been done, the wheels will be set to the 250th wheel position in the list of 500. Note that, before **[Set for Colossus]** is pressed, the value **"500"** can be replaced with some other smaller or larger <u>even</u> number **(n)** up to the value **"65534"**. In this case, after the **n** wheel positions have been generated, the wheels will be set to the **(n/2)th** position.

Next, press the **[PR1]** button; so doing presets the Input box with a <u>pr</u>eset message; this message is currently fixed (hard-wired within the program) – a future release of the program may allow variation in this preset message. This fixed preset message takes the form:

```
"334488THIS9IS9MESSAGE9NO955WU883344"
```

This is to be understood as:

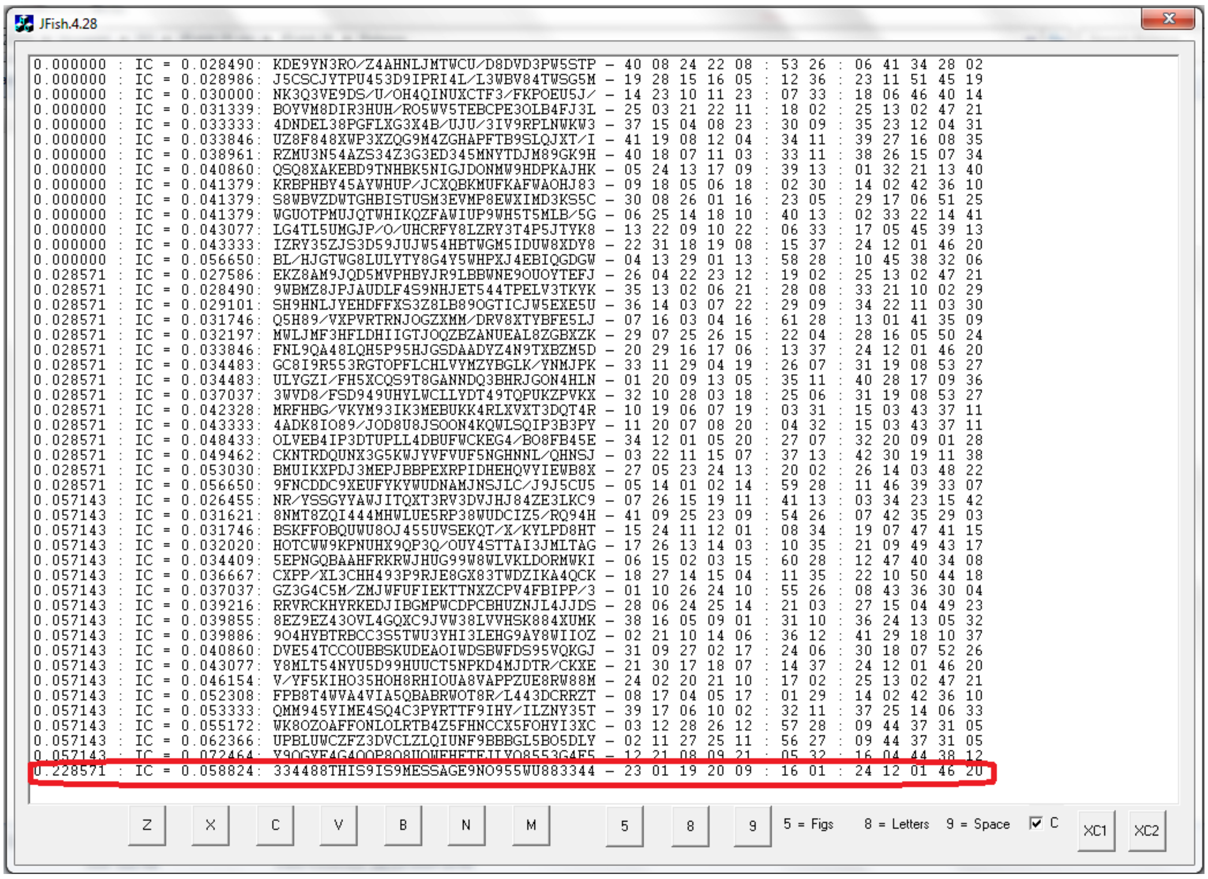
33	**<carriage return><carriage return>**
44	**<line feed><line feed>**
88	**<letter shift><letter shift>**
<text>	**THIS<spc>IS<spc>MESSAGE<spc>NO<spc>**
55	**<figure shift><figure shift>**
"27"	27
88	**<line feed><line feed>**
33	**<letter shift><letter shift>**
44	**<carriage return><carriage return>**

The message is designed to be particularly Colossus-friendly, and has a high **"Double Δ"**.

Once the message has been set, it is then encrypted.

Next, press the **[Colossus mode]** button; so doing uses the earlier generated 500 (or n) wheel positions to attempt the decryption of the previously encrypted pre-set message, sorting the results in order of the **"Double Δ"** value for each decryption.

If all goes well, the correct decryption, with its wheel settings will appear near, or ideally at, the bottom of the listing produced at the end of the run.



```
0.000000 : IC = 0.028490: KDE9YN3RO/Z4AHNLJMTWCU/D8DVD3PW5STP - 40 08 24 22 08 : 53 26 : 06 41 34 28 02
0.000000 : IC = 0.028986: J5CSCJYTPU453D9IPRI4L/L3WBV84TWSG5M - 19 28 15 16 05 : 12 36 : 23 11 51 45 19
0.000000 : IC = 0.030000: NK3Q3VE9DS/U/OH4QINUXCTF3/FKPOEU5J/ - 14 23 10 11 23 : 07 33 : 18 06 46 40 14
0.000000 : IC = 0.031339: BOYVM8DIR3HUH/RO5WV5TEBCPE3OLB4FJ3L - 25 03 21 22 11 : 18 02 : 25 13 02 47 21
0.000000 : IC = 0.033333: 4DNDEL38PGFLXG3X4B/UJU/3IV9RPLNWKW3 - 37 15 04 08 23 : 30 09 : 35 23 12 04 31
0.000000 : IC = 0.033846: UZ8F848XWP3XZQG9M4ZGHAPFTB9SLQJXT/I - 41 19 08 12 04 : 34 11 : 39 27 16 08 35
0.000000 : IC = 0.038961: RZMU3N54AZS34Z3G3ED345MNYTDJM89GK9H - 40 18 07 11 03 : 33 11 : 38 26 15 07 34
0.000000 : IC = 0.040860: QSQ8XAKEBD9TNHBK5NIGJDONMW9HDPKAJHK - 05 24 13 17 09 : 39 13 : 01 32 21 13 40
0.000000 : IC = 0.041379: KRBPHBY45AYWHUP/JCXQBKMUFKAFWAOHJ83 - 09 18 05 06 18 : 02 30 : 14 02 42 36 10
0.000000 : IC = 0.041379: S8WBVZDWTGHBISTUSM3EVMP8EWXIMD3KS5C - 30 08 26 01 16 : 23 05 : 29 17 06 51 25
0.000000 : IC = 0.041379: WGUOTPMUJQTWHIKQZFAWIUP9WH5T5MLB/5G - 06 25 14 18 10 : 40 13 : 02 33 22 14 41
0.000000 : IC = 0.043077: LG4TL5UMGJP/O/UHCRFY8LZRY3T4P5JTYK8 - 13 22 09 10 22 : 06 33 : 17 05 45 39 13
0.000000 : IC = 0.043333: IZRY35ZJS3D59JUJW54HBTWGM5IDUW8XDY8 - 22 31 18 19 08 : 15 37 : 24 12 01 46 20
0.000000 : IC = 0.056650: BL/HJGTWG8LULYTY8G4Y5WHPXJ4EBIQGDGW - 04 13 29 01 13 : 58 28 : 10 45 38 32 06
0.028571 : IC = 0.027586: EKZ8AM9JQD5MVPHBYJR9LBBWNE9OUOYTEFJ - 26 04 22 23 12 : 19 02 : 25 13 02 47 21
0.028571 : IC = 0.028490: 9WBMZ8JPJAUDLF4S9NHJET544TPELV3TKYK - 35 13 02 06 21 : 28 08 : 33 21 10 02 29
0.028571 : IC = 0.029101: SH9HNLJYEHDFFXS3Z8LB89OGTICJW5EXE5U - 36 14 03 07 22 : 29 09 : 34 22 11 03 30
0.028571 : IC = 0.031746: Q5H89/VXPVRTRNJOGZXMM/DRV8XTYBFE5LJ - 07 16 03 04 16 : 61 28 : 13 01 41 35 09
0.028571 : IC = 0.032197: MWLJMF3HFLDHIIGTJOQZBZANUEAL8ZGBXZK - 29 07 25 26 15 : 22 04 : 28 16 05 50 24
0.028571 : IC = 0.033846: FNL9QA48LQH5P95HJGSDAADYZ4N9TXBZM5D - 20 29 16 17 06 : 13 37 : 24 12 01 46 20
0.028571 : IC = 0.034483: GC8I9R553RGTOPFLCHLVYMZYBGLK/YNMJPK - 33 11 29 04 19 : 26 07 : 31 19 08 53 27
0.028571 : IC = 0.034483: ULYGZI/FH5XCQS9T8GANNDQ3BHRJGON4HLN - 01 20 09 13 05 : 35 11 : 40 28 17 09 36
0.028571 : IC = 0.037037: 3WVD8/FSD949UHYLWCLLYDT49TQPUKZPVKX - 32 10 28 03 18 : 25 06 : 31 19 08 53 27
0.028571 : IC = 0.042328: MRFHBG/VKYM93IK3MEBUKK4RLXVXT3DQT4R - 10 19 06 07 19 : 03 31 : 15 03 43 37 11
0.028571 : IC = 0.043333: 4ADK8IO89/JOD8U8JSOON4KQWLSQIP3B3PY - 11 20 07 08 20 : 04 32 : 15 03 43 37 11
0.028571 : IC = 0.048433: OLVEB4IP3DTUPLL4DBUFWCKEG4/BO8FB45E - 34 12 01 05 20 : 27 07 : 32 20 09 01 28
0.028571 : IC = 0.049462: CKNTRDQUNX3G5KWJYVFVUF5NGHNNL/QHNSJ - 03 22 11 15 07 : 37 13 : 42 30 19 11 38
0.028571 : IC = 0.053030: BMUIKXPDJ3MEPJBBPEXRPIDHEHQVYIEWB8X - 27 05 23 24 13 : 20 02 : 26 14 03 48 22
0.028571 : IC = 0.056650: 9FNCDDC9XEUFYKYWUDNAMJNSJLC/J9J5CU5 - 05 14 01 02 14 : 59 28 : 11 46 39 33 07
0.057143 : IC = 0.026455: NR/YSSGYYAWJITQXT3RV3DVJHJ84ZE3LKC9 - 07 26 15 19 11 : 41 13 : 03 34 23 15 42
0.057143 : IC = 0.031621: 8NMT8ZQI444MHWLUE5RP38WUDCIZ5/RQ94H - 41 09 25 23 09 : 54 26 : 07 42 35 29 03
0.057143 : IC = 0.031746: BSKFFOBQUWU8OJ455UVSEKQT/X/KYLPD8HT - 15 24 11 12 01 : 08 34 : 19 07 47 41 15
0.057143 : IC = 0.032020: HOTCWW9KPNUHX9QP3Q/OUY4STTAI3JMLTAG - 17 26 13 14 03 : 10 35 : 21 09 49 43 17
0.057143 : IC = 0.034409: 5EPNGQBAAHFRKRWJHUG99W8WLVKLDORMWKI - 06 15 02 03 15 : 60 28 : 12 47 40 34 08
0.057143 : IC = 0.036667: CXPP/XL3CHH493P9RJE8GX83TWDZIKA4QCK - 18 27 14 15 04 : 11 35 : 22 10 50 44 18
0.057143 : IC = 0.037037: GZ3G4C5M/ZMJWFUFIEKTTNXZCPV4FBIPP/3 - 01 10 26 24 10 : 55 26 : 08 43 36 30 04
0.057143 : IC = 0.039216: RRVRCKHYRKEDJIBGMPWCDPCBHUZNJL4JJDS - 28 06 24 25 14 : 21 03 : 27 15 04 49 23
0.057143 : IC = 0.039855: 8EZ9EZ43OVL4GQXC9JVW38LVVHSK884XUMK - 38 16 05 09 01 : 31 10 : 36 24 13 05 32
0.057143 : IC = 0.039886: 9O4HYBTRBCC3S5TWU3YHI3LEHG9AY8WIIOZ - 02 21 10 14 06 : 36 12 : 41 29 18 10 37
0.057143 : IC = 0.040860: DVE54TCCOUBBSKUDEAOIWDSBWFDS95VQKGJ - 31 09 27 02 17 : 24 06 : 30 18 07 52 26
0.057143 : IC = 0.043077: Y8MLT54NYU5D99HUUCT5NPKD4MJDTR/CKXE - 21 30 17 18 07 : 14 37 : 24 12 01 46 20
0.057143 : IC = 0.046154: V/YF5KIHO35HOH8RHIOUA8VAPPZUE8RW88M - 24 02 20 21 10 : 17 02 : 25 13 02 47 21
0.057143 : IC = 0.052308: FPB8T4WVA4VIA5QBABRWOT8R/L443DCRRZT - 08 17 04 05 17 : 01 29 : 14 02 42 36 10
0.057143 : IC = 0.053333: QMM945YIME4SQ4C3PYRTTF9IHY/ILZNY35T - 39 17 06 10 02 : 32 11 : 37 25 14 06 33
0.057143 : IC = 0.055172: WK8OZOAFFONLOLRTB4Z5FHNCCX5FOHYI3XC - 03 12 28 26 12 : 57 28 : 09 44 37 31 05
0.057143 : IC = 0.062366: UPBLUWCZFZ3DVCLZLQIUNF9BBBGL5BO5DLY - 02 11 27 25 11 : 56 27 : 09 44 37 31 05
0.057143 : IC = 0.072464: Y9QGYF4G4QQP8Q8UQWFHFTFJLYQ8553G4F5 - 12 21 08 09 21 : 05 32 : 16 04 44 38 12
0.228571 : IC = 0.058824: 334488THIS9IS9MESSAGE9NO955WU883344 - 23 01 19 20 09 : 16 01 : 24 12 01 46 20
```

The buttons **[XC1]** and **[XC2]** are used, respectively to:

[XC1] To clear the list, and render it invisible, and to:
[XC2] Render the list invisible without clearing it, so that subsequent runs will be merged with any previous runs.

Finally, the **[Colossus Loop]** button is a macro button, which carries out, in sequence,the functionality of the **[Load Key]**, **[Random]** (incorporating **[Set Wheels]**), **[Set for Colossus]**, **[PR1]** and **[Colossus Mode]** buttons.

PART III – SCRIPTING.

The program *JFishStepper.exe*, which is to be found in the *Release* folder of **[2]**, can be used to automate the operation of the JFish SZ42 simulator. It may be useful for display and educational purposes....

Its function is to read a script "*JFishScript.txt*", previously copied to the folder containing the executable, byte by byte, and to send each character, preceded by the "**Alt**" key code, and followed by the "**Alt-release**" key code, to whichever program has the current focus.

The permitted characters are the numerals **{1 ... 9}** and the letters **{A ... Z}**. **<space>** is also permitted, and it converted to **'9'**.

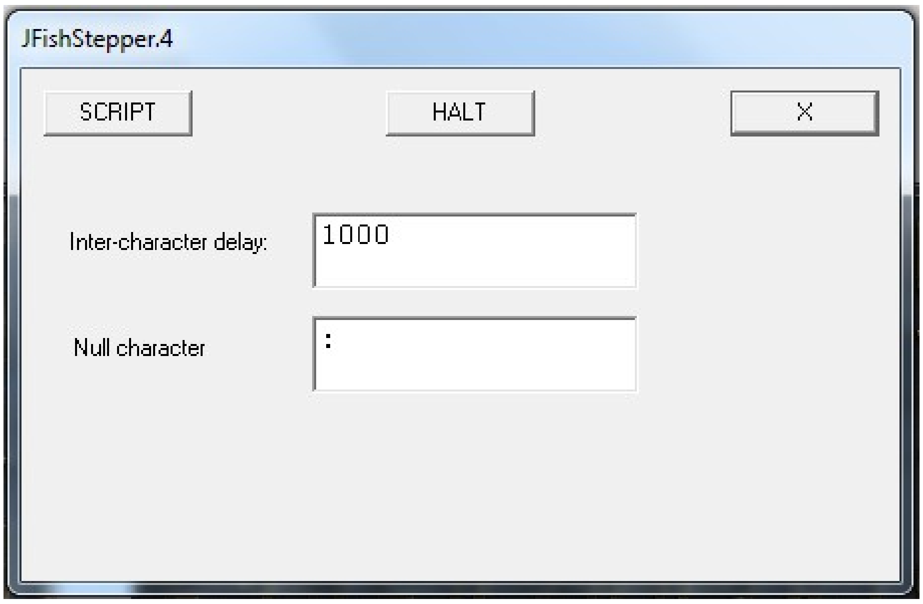
An example content of "**JFishScript.txt**" is this:

```
12WELCOME9TO9THE9SECOND9BIENNIAL::::
12CRYPTOSYMPOSIUM IN::::
12CHARLOTTE NORTH CAROLINA:::::
```

In this case, all characters are valid input characters to the SZ42 simulator, with the exception of the ':' (colon) character. The characters are sent at 1000 millisecond intervals, unless that interval is altered by the user. The ':' (colon) character represents a pause of the standard character interval.

So, in the given example, the characters **1, 2, W, E,** etc, are sent at 1000 msec intervals; at the end of each line there is are four colons, giving a pause of 4000 msec.

When *JFishStepper.exe* is started, by double-clicking on the executable of that name in the *Release* folder, the first screen that is shown is:



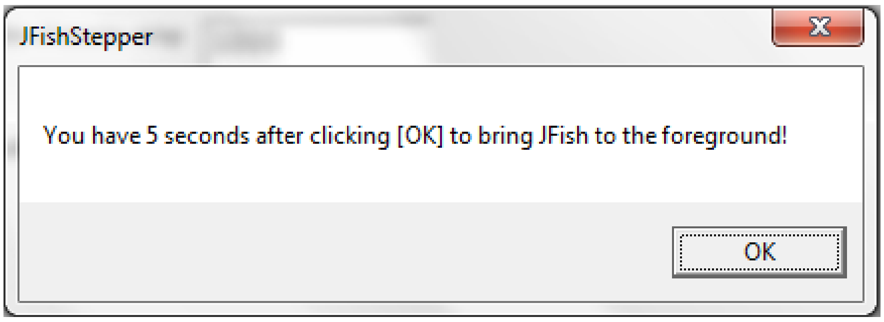
Inter-character delay: 1000

Null character :

The inter-character delay is preset to 1000 milliseconds, but can be altered to something else. It is not recommended to reduce its value below around 500 msecs as nothing will be seen on the SZ42 screen. Longer values are permitted as required.

The null character is preset to ':'(colon) but can be changed to any other printable character.

Pressing the **[SCRIPT]** key, or using its hot-key, **Alt+S**, results in the pop-up:



JFishStepper

You have 5 seconds after clicking [OK] to bring JFish to the foreground!

This basically means what it says! Press the **[OK]** button, and then, within 5 seconds, bring the SZ42 simulator to the foreground. It is best to have loaded the SZ42 simulator before starting ***JFishStepper***.

JFishStepper, once started, runs for ever. The simplest way to stop it is to bring it to the foreground, and click on the [HALT] button.

Appendices.

Appendix A.

The keyfile ***"JFishKey.txt"***:

```
...xxxx....xx....x.xx..x..xx.x.xx.x.xxxx.
xx..xxx.xx...x.x.xx...x....xxx.
..xxx.xx..x....xxx..xx.xx..xx
..xx..x.xx..x..xx..x..xxxx
.x...x.xx..x...xxx.xxx.
xxx.x.xx..xx..xx...xxxx.x.xx.xx...xx....xxxx.xx..xx...xx....x
x.xxx.x.x.x.x..x.x.xxx.x.x.x.x.x.x.x.
..x.x.x.x.x..x..x.xx.xx.x.x..xx.xxx..xxx...
..x.xx.x.x.x.x.x.xx..xx.x..x.xxxx.....xxx..x.xx
x.x.x.x.x.x.x..x..xx.x.x.xxxx....xxx...xxx.xx..x..x
x.x..xx.x.x.x.x.x.xx.x....xx..xx..xx.xxxxx.x..x....x.
.x.x.x.x.xx...x.x..xxx.xxxx.xx.x....x...x..xx.xx..xx..x.x.x
 1 1 1 1 1  1 1  1 1 1 1 1  LIM_CHI2
```

This is a file of 13 lines; each of the first 12 lines gives the pin settings for the 12 wheels (the K, M, and S wheels in order).

The thirteenth line gives the starting wheel positions for those wheels. Note that there may be a text after the 12th position indicating some kind of limitation setting. This is intended for a possible future enhancement; at the moment, no limitation settings are implemented and this text is ignored.

Appendix B

The Baudot (5-track teleprinter) code.

						'L'	'F'								'L'	'F'
			.	O	O	A			O		O	.	O		P	0
O	O		.		O	B	?		O		O	.	O	O	Q	1
	O	O	.	O		C	:			O		.	O		R	4
	O		.		O	D					O	.		O	S	'
			.		O	E	3		O			.			T	5
	O	O	.		O	F	!				O	.	O	O	U	7
O	O		.	O		G	&		O	O	O	.	O		V	;
O		O	.			H	£		O			.	O	O	W	2
		O	.	O		I	8		O	O	O	.		O	X	/
	O		.	O	O	J	Bell		O		O	.		O	Y	6
	O	O	.	O	O	K	(		O			.		O	Z	+
O			.	O		L	)		O	O	O	.	O	O	->L	->L
O	O	O	.			M	.		O	O		.	O	O	->F	->F
	O	O	.			N	,			O		.			CR	CR
O	O		.			O	9					.	O		LF	LF
											O	.			spc	spc

Appendix C.

“The Index of Coincidence” (“IC”) is defined as:

$$\frac{\Sigma(f_i * (f_i - 1))}{N(N-1)}$$

where:

i is in the range 1 to 26*, and represents the number of unique letters in the sample,

f*i* is the number of occurrences of the *i*th letter of the alphabet in the sample, and,

N is the total count of the letters in the sample.

** or whatever the alphabet size might be.*

The **IC** calculates the non-randomness of a string of letters; for a given piece of text, in a given language, the **“IC”** tends towards a specific value for that language. Random text, such as cryptotext, other than trivial examples thereof such as letter-for-letter substitution, is likely to have the most random distribution and will be likely to have the a value lower than that for any plain-text.

Appendix D - The hot-keys.

Most, but not quite all, of the clickable buttons / tick boxes can be operated from the PC keyboard. This can be done by pressing the **Alt** key on the keyboard, followed by the key of interest, followed by releasing the **Alt** key.

It is possible to hold down the **Alt** key continuously for many key inputs, but it is necessary to ensure that Windows doesn't interrupt the operation with its "sticky keys" pop-ups!

The buttons with hot-keys are:

all the buttons on the simulator keyboard, that is letters **{A | B | ... | Z}**,
numbers/characters **{/ | 3 | 4 | 5 | 8 | 9}**.
These buttons have **ALT** + {<*letter*> | <*number*> | /} respectively.

The **[RotateMachine]** button (which has **Alt + 0**)
The **[Load Key]** button (which has **Alt + 1**)
The **[Set Wheels]** button (which has **Alt + 2**)
The **[Random]** button (which has **Alt + 6**)
The **[X]** (= exit simulator) (which has **Alt + 7**)
The two **[TS]** (= transfer to storage) and **[AUTO]** buttons (which have
ALT + . (*fullstop or period*), ALT + **,** (*comma*) and **ALT + ;** (*semicolon*)
respectively.

*** End of Document ***