|  |  |
| --- | --- |
| TEA PRIMITIVE | SEMANTICS |
| Z: | |  |  | | --- | --- | | NAME | Zap | | PURPOSE | Do things in TEA using external power | | SYNTAX  & SEMANTICS | |  | | --- | | z: | | Return AI transformed to ALL LOWERCASE | | z:CMD | | Invoke the system command CMD passing AI as only input, set the result as IO iff the command successfully executed, otherwise return EMPTY STRING  On the WEB/WEB TEA, CMD is any valid code or string that can be evaluated in JavaScript using eval, and the AI is passed as “var AI=’ | | z!: | | Return AI transformed to ALL UPPERCASE | | z!:CMD | | Same as z:CMD, but will return an Error Message from the command execution if it didn’t execute or if there were errors. | | z\*: | | Return AI transformed to TITLE CASE | | z\*:vCMD  z\*!:vCMD | | Same as z:CMD and z!:CMD, but using the command string stored in the vault with name vCMD | |  | | NOTES | The Z-command space is for bringing external (think Unix, System or non-TEA) powers into a TEA program… But also, much as its outputs might be harder to predict or determine up-front, and yet, among the TEA primitives, it is one of the simplest to use and recall, because of the simplicity of its API.  Also, because the Z: primitive allows a TEA program to set the value of the ACTIVE input to whatever that command returns from executing an external system command, it then becomes a potential way to read external values into the TEA program, and also might allow writing the ACTIVE input to some external memory store. Other input commands such as **i:** and **w:** can be likened to **z:** in this regard, with the exception that any potential data or values that might be introduced into a TEA program using the **i:** primitive is explicitly stated in the TEA program code.  The earlier sections of this specification already contain several example TEA programs using the Z-primitive. But, perhaps, it should be noted, that, just like the network accessing commands in the W-command space might not do what they are expected to do when the program is being run without access to a network – such as when a program meant to post data to a server runs while the host device has no data connection, also, in some operating environments – such as when a TEA program is being run in an environment that doesn’t expose the system – such as on un-rooted mobile devices – they might be running on Linux underneath, but this space might not accessible to TEA programs by default. While, in other environments like Windows or LINUX, such access is guaranteed to be possible most times. Thus, Z-primitives should be used cautiously, and perhaps only minimally or not at all if possible, so as to ensure that one’s TEA programs shall do what they are expected to do portably or across all environments. Z-command space should mostly be kept for advanced, non-trivial cases that are expected to only be required in special environments one is sure about.  Z: basically makes it possible to perform some basic system programming in TEA. For a simple example demonstrating the power of the Zap facility, here is a program that would generate random numbers from words:  i:abcdefghijklmnopqrstuvwxy z  a!:  d:[ ].\*$  r!:[aeiou]:0  r!:[bcdfghjklmnpqrstvwxyz]:1  x:{ibase=2; }  x:{echo '}  x!:{' | bc}  v:vCMD  c:  z\*:vCMD  Where we assume the system or shell command `bc` exists. Also, this example shows how we can use TEA to construct custom or special system commands, and then use them to solve problems based on user-provided data. For example this program constructs a call such as  echo 'ibase=2; 1100011011111101111' | bc  Which, when passed to the Z-command returns “407535” on a system where the “bc” command is accessible. A slightly modified version of this program is shown below, which would take any user input and return it as a number. It can for example show that the name “Jesus Christ” is equal to the number 1403!  # program converts provided word to a number  i:test  g!: # first eliminate all whitespace  z: # then turn everything to lowercase  r!:[aeiou]:0 | r!:[bcdfghjklmnpqrstvwxyz]:1  x:{ibase=2; } | x:{echo '} | x!:{' | bc}  v:vCMD | c: | z\*:vCMD  # Magic!    An otherwise simple, and likely potentially widely supported system command is the “pwd” command for returning the current working directory. We can see Z: working with an example invoking this command via TEA:  EXPERIMENTS|< 18:26:34 $>\* pwd  /mnt/e/LAB/EXPERIMENTS  EXPERIMENTS|< 18:26:36 $>\* tttt -c "z:pwd"  /mnt/e/LAB/EXPERIMENTS  EXPERIMENTS|< 18:26:50 $>\*  Another useful demonstration is with combining use of TEA vaults with the Z-utility, such as in this example:  i:{+'%A %d, %b %Y'}|v:vCMD:{date}|z\*:vCMD  Which should return something like "Friday 23, Aug 2024"  For those testing Z: via TEA on the WEB (where we can’t readily run the same stuff we can run in Z on the command-line TEA, since there is not bash, no Linux/Unix commands), examples for using Z: to run JavaScript from TEA are as follows:  i!:2|z:AI+3  That shall return “23” since, as we can see in the TEA debug output for this call, the system sets the internal variable AI to a string value “2” based on the AI at the time Z: is called, and so, whatever we write in the CMD against AI refers to the string value, unless we marshal it manually. See the debug output below for illustration:  Processing Instruction: z:AI+3  PRIOR MEMORY STATE: (=2, VAULTS:{})  \*\*\*[SYSTEM CMD]: EVALUATING via JavaScript:  ---[START CMD]---  const AI = "2"; AI+3  ---[END CMD]---  \*\*\*[SYSTEM CMD RESULT]: 23  RESULTANT MEMORY STATE: (=23, VAULTS:{})  So, to instead treat it as an integer and thus return say “5”, we should re-write that program as:  i!:2|z:Number(AI)+3 #=5  The debug output shall enlighten us:  Processing Instruction: z:Number(AI)+3  PRIOR MEMORY STATE: (=2, VAULTS:{})  \*\*\*[SYSTEM CMD]: EVALUATING via JavaScript:  ---[START CMD]---  const AI = "2"; Number(AI)+3  ---[END CMD]---  \*\*\*[SYSTEM CMD RESULT]: 5  RESULTANT MEMORY STATE: (=5, VAULTS:{})  Running invalid JavaScript or if any exception occurs during evaluation of that CMD, then with cases like Z!:, we would get the result containing the error:  i:2|z:Numbers(AI)+3 # (=, VAULTS:{})  i:2|z:Numbers(AI)+3 # (=[ERROR]: ReferenceError: Numbers is not defined, VAULTS:{})  And for the case of commands held in vaults, we see the following as a good example:  i:2|v:vCMD:Number(AI)+3|z\*:vCMD # (=5, VAULTS:{"vCMD":"Number(AI)+3"})  **WARNING**: BE CAREFUL with the POWER of Z: --- not just on WEB, but anywhere! Take for example the following simple program that, when run on the TEA WEB IDE, shall modify the WEB IDE itself! Yes, it shall insert the text “Hello Word 5” somewhere towards the bottom of the page in RED, and after that, unless you reload the page, most things will just stop working!  **i:2|v:vCMD:{var R=Number(AI)+3;document.body.innerHTML += "<div style=\'color:red;\'>Hello <b>World:" + R +"</b></div>";R}|z\*:vCMD** |  |  | |