|  |  |
| --- | --- |
| TEA PRIMITIVE | SEMANTICS |
| W: | |  |  | | --- | --- | | NAME | Webify | | PURPOSE | Read or Write to the Network | | SYNTAX  & SEMANTICS | |  | | --- | | w: | | Take current AI as a URL and read whatever is at specified network resource address with an HTTP GET request, then set the result as IO iff the command successfully executed, otherwise return EMPTY STRING | | w:URL | | Read whatever is at specified network resource address URL and set the result as IO iff the command successfully executed, otherwise return EMPTY STRING | | w!: | | Take current AI as a URL and read whatever is at specified network resource address with an HTTP POST request, then set the result as IO iff the command successfully executed, otherwise return EMPTY STRING | | w!:URL | | Write or rather, POST whatever is in AI, to the specified network resource address URL as unnamed data and set the result as IO iff the command successfully executed, otherwise return EMPTY STRING | | w\*:  w\*:vURL  w\*:vURL:v1:v2:v:3..:vN | | Same as w!:URL but will execute an HTTP GET request on the URL held in vault vURL with each specified vault (as vNAME=vVALUE pairs) in the query string. With only vURL, posts everything currently in the vaults, using their name and values. With no parameters at all, treats AI as URL, then sends everything in the vaults in the query string. | | w\*!:  w\*!:vURL  w\*!:vURL:v1:v2:v:3..:vN | | Same as W\*:, but performs an HTTP POST request and posts the data as form encoded data | |  | | NOTES | TEA might not offer any usual means to read or write files such as many programming languages provide, however, it does offer a clean solution to reading and writing data to a network-accessible resource---which, given it can be purely offline or on-device, allows for a sort of File I/O in TEA via the W: command space.  Perhaps to clarify things – when a TEA program reads using W:URL, it is expected that the body of the returned response – such as an HTML document for web requests, or perhaps a data dump in JSON, CSV or such, for an API call, is what is returned by W:. In cases where an error occurs or the resource doesn’t exist, W: just returns an empty string.  For writing/posting data, W! and W\*!: come in handy. The first version posts AI just as a mere untagged value – just like posting a string value to some API end-point for example. The vault-accessing method though, because it has access to both the names and values in the vaults, will compose a sane multi-part request, where each vault and its value (or the empty string if no value) gets posted to the specified URL. HTTP POST requests are the default for W!\* because where HTTP GET is desired, the necessary URL with the necessary name-value pairs in a query-string can be composed using TEA, but is not encouraged because of its lack of security, and potential to construct illegal URLs given arbitrary data.  The following example will fetch a person’s current account balance from a bank’s API given the account number and some special secret.  V:vAC:{123456XXX} | v:vKEY:{001001}  w\*!: https://abc.bank/api/ac/bal:vAC:vKEY  # (= “321,000,000,000”)  However, away from hypothetical bank APIs, the above example can be tested using the HTTP echo API service as such:  V:vAC:{123456XXX} | v:vKEY:{001001}  v:vURL:http://httpbin.org/post  w\*!:vURL:vAC:vKEY  And this returns a result as the following (via the TEA WEB IDE)…  {  "args": {},  "data": "",  "files": {},  "form": {  "vAC": "123456XXX",  "vKEY": "001001"  },  "headers": {  "Accept": "\*/\*",  "Accept-Encoding": "gzip, deflate",  "Accept-Language": "en-US,en;q=0.9",  "Content-Length": "25",  "Content-Type": "application/x-www-form-urlencoded",  "Host": "httpbin.org",  "Origin": "http://localhost",  "Referer": "http://localhost/",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36",  "X-Amzn-Trace-Id": "Root=1-68cea643-24a04d4e2f3d5d41463a2474"  },  "json": null,  "origin": "41.210.143.35",  "url": "http://httpbin.org/post"  }  A modified version that only specifies the vault from which to read the URL to which to post to, shall help illustrate the fact that W\*!: shall post everything in the vaults, if they weren’t explicitly specified, thus, the code:  V:vAC:{123456XXX} | v:vKEY:{001001} | v:vTEST:{Hello World}  v:vURL:http://httpbin.org/post  w\*!:vURL  Shall produce the following output:  {  "args": {},  "data": "",  "files": {},  "form": {  "vAC": "123456XXX",  "vKEY": "001001",  "vTEST": "Hello World",  "vURL": "http://httpbin.org/post"  },  "headers": {  "Accept": "\*/\*",  "Accept-Encoding": "gzip, deflate",  "Accept-Language": "en-US,en;q=0.9",  "Content-Length": "80",  "Content-Type": "application/x-www-form-urlencoded",  "Host": "httpbin.org",  "Origin": "http://localhost",  "Referer": "http://localhost/",  "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36",  "X-Amzn-Trace-Id": "Root=1-68cea775-1ea4aef3738a6c3c11d9f66c"  },  "json": null,  "origin": "41.210.143.35",  "url": "http://httpbin.org/post"  }  So, that version needs be used with precaution to not accidentally expose secret or private data to the remote server! Actually, precaution needs be taken with all W: calls in this regard.  That’s with **W\*!:** which does an HTTP POST of data in the vaults.  The following example demonstrates this facility using the interesting **W\*:** that knows to post everything in the vaults via HTTP GET automatically!  v:A:123|v:vTest:{some value}|i:http://httpbin.org/get|w\*:  returns the result:  {  "args": {  "A": "123",  "vTest": "some value"  },  "headers": {  "Accept-Encoding": "identity",  "Host": "httpbin.org",  "User-Agent": "Python-urllib/3.10",  "X-Amzn-Trace-Id": "Root=1-66c83059-040326d238fdaff202c56b4c"  },  "origin": "41.210.147.213",  "url": "http://httpbin.org/get?A=123&vTest=some+value"  }    Concerning the vault-less versions of W:, we shall only look at one example and then move on.  i!:https://freeipapi.com/api/json/|w:  That TEA program essentially just takes the AI when W: is invoked, uses it as the URL and performs an HTTP GET to that endpoint, returning the result as IO  On the commandline/with CLI TEA/tttt, we see this work out as such:  tttt -c "i!:https://freeipapi.com/api/json/|w:"  {"ipVersion":4,"ipAddress":"41.210.143.35","latitude":-0.07722000000000001,"longitude":31.4567,"countryName":"Uganda","countryCode":"UG","capital":"Kampala","phoneCodes":[256],"timeZones":["Africa\/Kampala"],"zipCode":"-","cityName":"Sembabule","regionName":"Central Region","continent":"Africa","continentCode":"AF","currencies":["UGX"],"languages":["en","sw"],"asn":"20294","asnOrganization":"MTN Uganda","isProxy":false}  However, and this is important to note concerning HTTP calls to various endpoints – sometimes, some endpoints either block some particular ORIGIN DOMAINs or CLIENTs. And thus, the same code above, when run on the WEB TEA instead results in the following error:  [ERROR]: NetworkError: Failed to execute 'send' on 'XMLHttpRequest': Failed to load 'https://freeipapi.com/api/json/'.  Moreover, currently TEA’s WEB functions attempt to perform synchronous HTTP calls first of all, but also, TEA doesn’t yet allow the user to modify or set any special HTTP request headers when using the W: utilities. More research and experimentation is due concerning how best to implement/design this TEA instruction. Stay on the look-out for future advances… |  |  | |