|  |  |
| --- | --- |
| TEA PRIMITIVE | SEMANTICS |
| L: | |  |  | | --- | --- | | NAME | Label | | PURPOSE | Explicitly mark sections in a TEA Program accessible by unique names | | SYNTAX  & SEMANTICS | |  | | --- | | L: | | INERT | | L:LABEL | | Declare a jump position in a TEA program accessible by the name LABEL | | L!: | | INERT | | L!:LABEL  L!:LABEL1:LABEL2:LABEL3:..:LABELN | | Similar to L:, but allows for the same position in the program to be accessible using any of the specified label names or tags, which are expected to be unique not only across the list, but also across the entire TEA Program. | |  | | NOTES | Note that for the overloadable TEA Label construct **L!:,** any branching command that references any of the label values in the label set expression for a particular **l!:** in the TEA program, makes the program jump there irrespective of which of the names was used. This allows TEA programs to implement useful ideas such as functions, polymorphism, more humane-APIs, etc.  Also, note that, to emphasize this important TEA feature, only the Inverse Label construct allows overloading, much as both the **l:LABEL** and **l!:LABEL** constructs work the same for singular labels. The following is a non-trivial example of how labelling solves problems: in TEA programming:  # TEA data processing prog with some kind of error handling  v:vLOG:{--No Processing Yet--}  l:FETCH  w!: <https://pastebin.org/KYC.csv> # rqrd 2b sme non-empty result  f!:^$:PROCESS:ERROR | # process iff data is not empty  l:PROCESS  v:vDATA  z!: date -Ins | v:vDATE  # now combine current date & logs with the data  g\*:{---}:vDATE:vLOG | q!: | # end by returning data report  l:ERROR  | v:vERROR:{--Data Access Error--}  g\*:{\*\*\*}:vLOG:vERROR | v:vLOG | # update the log  j:FETCH | # then re-try the data processing |  | |