Coverage Report

Watched: 172 within `#.Pakkman.Registry, #.Pakkman.Server, #.Pakkman.Client`

90 of them are 100% covered

47 are not covered at all:

```
#.Pakkman.Client.CreateAllRefs_Client
#.Pakkman.Client.GetMyUCMDsFolder
#.Pakkman.Client.GetUserHomeFolder
#.Pakkman.Client.Help
#.Pakkman.Client.HelpOnSyntax
#.Pakkman.Client.InitUserSettings
#.Pakkman.Client.LF
#.Pakkman.Client.ListVersions_
#.Pakkman.Client.LoadDependencies
#.Pakkman.Client.PublishPackageToRemoteRegistry__
#.Pakkman.Client.Version
#.Pakkman.Client.UserSettings.make0
#.Pakkman.Client.UserSettings.make2
#.Pakkman.Client.Dependencies.ReduceDependencyPath
#.Pakkman.Server.CR
#.Pakkman.Server.CopyDependencies
#.Pakkman.Server.CreateHomePage
#.Pakkman.Server.FetchDescriptionAndPackageProjectUrl
#.Pakkman.Server.GetBestVersion
#.Pakkman.Server.GetDetails
#.Pakkman.Server.GetFooter
#.Pakkman.Server.GetGroupList
#.Pakkman.Server.GetHomeFolder
#.Pakkman.Server.GetPackageConfigFilename
#.Pakkman.Server.GetRegistryFolder
#.Pakkman.Server.GetVersionList
#.Pakkman.Server.HtmlHeader
#.Pakkman.Server.JSON
#.Pakkman.Server.NL
#.Pakkman.Server.PROJECT_HOME
#.Pakkman.Server.ParsePackageName
#.Pakkman.Server.PrepareHtmlPageForGroupList
#.Pakkman.Server.PrepareHtmlPageForPackageDetails
#.Pakkman.Server.PrepareHtmlPageForPackageList
#.Pakkman.Server.PrepareHtmlPageForVersionList
#.Pakkman.Server.QuitDQ
#.Pakkman.Server.ReturnFiles
#.Pakkman.Server.ReturnHTML
#.Pakkman.Server.ReturnNOT_IMPLEMENTED
#.Pakkman.Server.Run
#.Pakkman.Server.Timestamp
#.Pakkman.Server.ΔYesOrNo
#.Pakkman.Registry.Create_UUID
#.Pakkman.Registry.IsRegistry
#.Pakkman.Registry.Latestversions
#.Pakkman.Registry.OperatingSystem
#.Pakkman.Registry.Version
```

35 are partly covered:

Function/Operator	Lines not executed	Coverage	∑*1
#.Pakkman.Client.LoadPackage	27 ∑=1	3%	33
#.Pakkman.Client.InstallPackage	19 ∑=1	4%	26
#.Pakkman.Client.GetDependecyTreeViaHTTP	7 ∑=1	5%	21
#.Pakkman.Client.LinkTargetToPackage	21 ∑=1	6%	17
#.Pakkman.Client.UserSettings.AddRegistry	18,19 ∑=2	6%	34
#.Pakkman.Client.UserSettings.ReplaceRegistry	11 ∑=1	7%	14
#.Pakkman.Registry.ListPackages	20 ∑=1	7%	14
#.Pakkman.Client.UserSettings.RemoveRegistry	9 ∑=1	8%	12
#.Pakkman.Server.GetBestVersionNumber	15 ∑=1	8%	12
#.Pakkman.Server.OnRequest	12 ∑=1	11%	9
#.Pakkman.Client.GetBestVersionNumber	4,16 ∑=2	13%	15
#.Pakkman.Client.GetPackageViaHTTP_	8 ∑=1	14%	7
#.Pakkman.Client.UserSettings.GetRegistry	6 ∑=1	14%	7
#.Pakkman.Registry.Index.FindPartPackageID	9 ∑=1	14%	7
#.Pakkman.Client.GetDependecyTree	54,55,56,57,59,60,61 ∑=7	15%	48
#.Pakkman.Registry.Index.FindFullPackageID	8 ∑=1	17%	6
#.Pakkman.Server.Handle_PUT	6 ∑=1	17%	6
#.Pakkman.Client.LoadBuildList	13,15 ∑=2	18%	11
#.Pakkman.Server.FetchPackage	2,14 ∑=2	20%	10
#.Pakkman.Server.GetDependencies	11,17 ∑=2	20%	10
#.Pakkman.Client.GetAllFiles	10,11,12,13 ∑=4	21%	19
#.Pakkman.Client.ScanRegistriesFor	13,14,28,29,30 ∑=5	22%	23
#.Pakkman.Client.GetMD5	6,7 ∑=2	25%	8
#.Pakkman.Client.ListVersions	3 ∑=1	25%	4
#.Pakkman.Server.Handle_GET	2,6,8,21 ∑=4	25%	16
#.Pakkman.Server.ProcessCredentials	6 ∑=1	25%	4
#.Pakkman.Server.ProcessTestCommands	10,11,17,19,22 ∑=5	28%	18
#.Pakkman.Client.GetPakkmanRootSpace	9,10,12 ∑=3	30%	10
#.Pakkman.Server.HandleREST_Version1	15,17,18,22,24,30,32,33 ∑=8	30%	27
#.Pakkman.Client.ValidatePackageFiles	4,5 ∑=2	33%	6
#.Pakkman.Registry.Base64	7,8,9 ∑=3	33%	9
#.Pakkman.Registry.PROJECT_HOME	5,6,8 ∑=3	43%	7
#.Pakkman.Server.OnHouseKeeping	3,4,5,6 ∑=4	67%	6
#.Pakkman.Server.GetPackageList	4,10,11,12,14,16,17,18,19,20,21,22,23,24 ∑=14	78%	18
#.Pakkman.Client.LoadInstalledDependencies	3,4,5,6,7,8,9,10,11,12,13 ∑=11	85%	13

*1: Total of testable lines: comment lines, empty lines, all `:End*` lines etc. are ignored.

Listings

#.Pakkman.Client.LoadPackage

```
 refs←LoadPackage y
[1]   ;buildList;url;level;packageID;tempDir;saveIn;cfg;msg;ref;tempDirs;reset;identifier
[2]  ⍝ Load package `identifier` dynamically into the workspace,\\
[3]  ⍝ Right argument `y` must be `(identifier_ targetSpace)`\\
[4]  ⍝ `identifier` must be one of:
[5]  ⍝ * an HTTP request for a package
[6]  ⍝ * a ZIP file holding a package
[7]  ⍝ * a folder holding a package (like file://C:/Temp/group-name-version)
[8]  ⍝ * a path to a package in a registry (like [RegistryAlias]{packageID} or C:\MyReg{packageID}')
[9]  ⍝ * a package ID; Pakkmen will then attempt to find that package in the Registries defined in Client's config fil
[10] ⍝ `targetSpace` must be a fully qualified name of an ordinary namespace, meaning it start with either `#.` or `⎕S
[11] ⍝ `targetSpace` might already exist but if it doesn't it will be created. If it exists but is not an ordinary nam
[12] ⍝ an error is thrown.\\
[13] ⍝ `←` Vector with all references that were loaded, including depencies.\\
[14] ⍝ Loads the package into `#._packages.{packageName}` and establishes a reference pointing to it in `targetSpace`\\
[15] ⍝ Loads all dependencies, if any, as well.\\
[16] ⍝ Leaves no trails in the file system unless a package relies on file assets in which case those are loaded into
[17] ⍝ temporary directory.\\
[18] ⍝ Returns references to all packages loaded into Pakkman's package namespace.
[19]  (identifier_ targetSpace)←y
[20]  refs←reset←tempDirs←⍬
[21]  identifier←ReplaceRegistryAlias identifier_
[22]  ('Unknown',(∧/'[]'∊identifier_)/' alias')Assert 0<≢identifier
[23]  targetSpace←⍕targetSpace
[24]  '"targetSpace" must start with either "#" or "⎕SE"'Assert(1 ⎕C{ω⍷⍨¯1↑ω⍳'.'}targetSpace)∊'#' '⎕SE'
[25]  saveIn←GetPakkmanRootSpace targetSpace
[26]  :If 0=⎕NC targetSpace
→[27]     targetSpace(⍎{ω⍷⍨¯1↑ω⍳'.'}saveIn).⎕NS''
[28]  :Else
```

```
 [29]      '"targetSpace" is not a namespace'Assert 9=⎕NC targetSpace
 [30]      '"targetSpace" must not be a scripted namespace'Assert{16::1 ⋄ 0⊣⎕SRC ⍵}⍎targetSpace
 [31]  :EndIf
 [32]  ('Notfound: ',identifier)Assert 0<≠buildList←GetDependecyTree identifier
 [33]  :For level packageID url :In ↓buildList
 [34]      tempDir←InstallPackage_ packageID url
 [35]      cfg←ReadPackageConfigFile tempDir
 [36]      packageID←1 GetPackageIDFromFilename url
 [37]      saveIn2←saveIn,'.',packageID
 [38]      msg←⎕SE.Link.Import saveIn2(tempDir,'/',cfg.source)
 [39]      'Link failed to import code'Assert'Imported:'{⍺≡(≠⍺)↑⍵}msg
 [40]      ref←⍎saveIn2
 [41]      ref.ΔURI←cfg.uri
 [42]      ref.ΔHOME←tempDir
 [43]      :If 0=≠cfg.files
 [44]          tempDirs,←⊂tempDir
 [45]          reset,←⍎saveIn2
 [46]      :EndIf
 [47]      refs,←ref
 [48]      refs,←targetSpace LoadInstalledDependencies tempDir
 [49]      cfg LinkTargetToPackage saveIn2 targetSpace
 [50]  :EndFor
 [51]  LinkDependencies refs
 [52]  :If 0<≠tempDirs
 [53]      F.RmDir tempDirs
 [54]      reset.ΔHOME←⊂''  ⍝ Those have been deleted because they don't have
 [55]  :EndIf
 [56] ⍝Done
```

#.Pakkman.Client.InstallPackage

```
  paths←InstallPackage(identifier targetFolder)
 [1]  ⍝ Install package `identifier` in `targetFolder`.\\
 [2]  ⍝ `identifier` must be one of:
 [3]  ⍝ * an HTTP request for a package
 [4]  ⍝ * a ZIP file holding a package
 [5]  ⍝ * a folder holding a package (like file://C:\Temp\group-name-version\\)
 [6]  ⍝ * a path to a package in a registry (like [RegistryAlias]{packageID} or C:\MyReg\{packageID}')
 [7]  ⍝ * a package ID; Pakkmen will then attempt to find that package in the Registries defined in the Client's config
 [8]  ⍝ `←` Paths to all packages loaded with the first one being the "main" package, meaning that all the others are d
 [9]   '"targetFolder" is invalid'Assert F.IsDir⍠targetFolder
 [10]  paths←⍬
 [11]  identifier←ReplaceRegistryAlias identifier
 [12]  id←{⍵↑⍨-{¯1+⌊/⍵⍳'/\'}⌽⍵}identifier
 [13]  :If ~Reg.IsValidPackageID_Complete id
 [14]      :If Reg.IsValidPackageID_WithMajorNo id
 [15]          id←GetBestVersionNumber id((-≠id)↓identifier)
 [16]      :ElseIf Reg.IsValidPackageID_WithoutVersionNo id
 [17]          id←GetBestVersionNumber id((-≠id)↓identifier)
 [18]      :Else
→[19]          'Invalid identifier'Assert 0
 [20]      :EndIf
 [21]      identifier←({⍵↓⍨-{⌊/⍵⍳'/\'}⌽⍵}identifier),'/',id
 [22]  :EndIf
 [23]  :If 0<≠buildList←GetDependecyTree identifier
 [24]      :For level packageID url :In ↓buildList
 [25]          installFolder←targetFolder,'/',packageID
 [26]          :If level=1
 [27]          :AndIf F.Exists installFolder
 [28]              ('Installation folder "',installFolder,'" is a file')Assert 0=F.IsFile installFolder
 [29]              ('Installation folder is not empty :"',installFolder,'"')Assert 0=≠⊃F.Dir installFolder,'/'
 [30]          :EndIf
 [31]          tempDir←InstallPackage_ packageID url
 [32]          installFolder ⎕NMOVE tempDir
 [33]          paths,←⊂installFolder
 [34]          :If 1=level
 [35]              targetFolder Dependencies.Add packageID
 [36]          :EndIf
 [37]          F.RmDir tempDir
 [38]      :EndFor
 [39]      buildList SaveBuildList targetFolder
 [40]  :EndIf
 [41]  ⍝Done
```

#.Pakkman.Client.GetDependecyTreeViaHTTP

```
  level GetDependecyTreeViaHTTP url
 [1]  ⍝ Fetch the dependency tree via HTTP. Once a package is retrieved via the http protocol
 [2]  ⍝ we know that all dependecies must be fetched via the http protocol as well.\\
 [3]  ⍝ Requires semi-global ΔTREE
 [4]   (uri packageID)←Reg.SplitAtLast url
 [5]   client←NewClient uri
 [6]   :If ':'∈RemoveHttpProtocol uri
→[7]      client.Port←⊃⊃(//)⎕VFI{⍵↓⍨⍵⍳':'}RemoveHttpProtocol uri
 [8]   :EndIf
 [9]   request←R.NewRequest 0
```

```
 [10]  request.URI←'/v1/packages/dependencies/',packageID
 [11]  request.Headers,←⊂'Accept' 'application/json'
 [12]  res←client R.SendAndReceive request
 [13]  Assert 200≡res.StatusCode
 [14]  :If 0<≢res.Content
 [15]  :AndIf 0<≢tree←{⍵.(LEVEL,[1.5]TREE)}⎕JSON⍠('Dialect' 'JSON5')⊣res.Content
 [16]      tree[;1]+←level
 [17]  :AndIf 0<≢tree←(~tree[;2]∈ΔTREE.{Data[;ΔpackageID]}⍬)⌿tree
 [18]  ⍝ Now dependencies on any server may point to another server or they won't mention a protocol at all,
 [19]  ⍝ but in that case they are on the very same server as to original package, so we inject the original url.
 [20]      isNotHttp←0=Reg.IsHTTP¨tree[;2]
 [21]      (isNotHttp⌿tree[;2])←(({⍵/¨3>:+\⍵='/'}url),'/')∘,¨isNotHttp⌿tree[;2]
 [22]      requiredBy←packageID
 [23]      ΔTREE.Data,←↑{⍵[1],(⊂{⍵↓¨⍨⍵ι'/'}RemoveHttpProtocol 2⊃⍵),⍵[2]}¨↓tree
 [24]      :If ∨/~isNotHttp
 [25]          (level+1)GetDependecyTreeViaHTTP¨(~isNotHttp)/tree[;2]
 [26]      :EndIf
 [27]  :EndIf
 [28] ⍝Done
```

#.Pakkman.Client.LinkTargetToPackage

```
  {r}←cfg LinkTargetToPackage(source target)
 [1]   r←⍬
 [2]   alias←cfg.alias{0=≢⍺:⍵ ⋄ ⍺}cfg.name
 [3]   name←cfg.name
 [4]   ref←⍎source
 [5]   :If 0<≢cfg.api
 [6]       from←cfg.api
 [7]   :Else
 [8]       buff←⎕NPARTS cfg.source
 [9]       :If buff[3]∈'.aplf' '.aplo'
 [10]          from←2⊃buff
 [11]      :Else
 [12]          from←cfg.name
 [13]      :EndIf
 [14]  :EndIf
 [15]  :If (ref.⎕NC from)∈3 4
 [16]      ⍝ If it is a function it must not be niladic!
 [17]      ('Cannot load package "',source,'": entry point is niladic function ')Assert 0≠2 1⊃ref.⎕AT from
 [18]  :EndIf
 [19]  :If 0=⎕NC source,'.',from
 [20]  :AndIf 0=|⎕NC⊂source,'.',from
→[21]      0 Assert¨⍨'Cannot establish API in ',source
 [22]  :EndIf
 [23]  target⍎alias,'←',source,'.',from
 [24] ⍝Done
```

#.Pakkman.Client.UserSettings.AddRegistry

```
  {r}←AddRegistry y
 [1]     ⍝ Adds the definition of a Registry (as a namespace) to the user settings.\\
 [2]     ⍝ If that Registry is already defined an error is thrown.\\
 [3]     ⍝ `⍵` must be one of:
 [4]     ⍝ * An instance of the "Registry" class.
 [5]     ⍝ * A text vector with a name or an alias+name in the format `[alias]name`\\
 [6]     ⍝   In this case `AddRegistry` creates an instance on the fly. Of course thies means that this is
 [7]     ⍝   only suitable in case specifying just the name and optionally the alias is sufficient.
 [8]     ⍝ * A namespace that can be used to instanciate the `DefineRegistry` class.\\
 [9]     ⍝ `←` is always `⍬` (shy).
 [10]  :Access Public Instance
 [11]  r←⍬
 [12]  'Invalid depth'Assert(≡registries)∈0 1
 [13]  :If 80=⎕DR y
 [14]      :If ∧/'[]'∈y
 [15]          (alias name)←{⍵{('[]'~¨⍨⍵↑⍺)(⍵↓⍺)}⍵ι']'}y
 [16]          alias←⎕C alias
 [17]      :Else
→[18]          name←y
→[19]          alias←''
 [20]      :EndIf
 [21]      ((name='\')/name)←'/'
 [22]      registry←⎕NEW home.DefineRegistry(,⊂name)
 [23]      registry.alias←⎕C alias
 [24]  :Else
 [25]      registry←y
 [26]  :EndIf
 [27]  '⍵ is not instance if the "DefineRegistry" class'Assert∨/'[DefineRegistry]'⍷⍕registry
 [28]  :If 0<≢_registries
 [29]      'This URI is already defined'Assert~(⊂registry.uri)∈_registries.uri
 [30]      :If 0<≢registry.alias
 [31]          'This alias is already used for a different uri'Assert~(⊂registry.alias)∈_registries.alias
 [32]      :EndIf
 [33]  :EndIf
 [34]  buff←registry.Get ⍬
 [35]  :If 0=buff.priority
```

```
 [36]      :If 0=≢_registries
 [37]          buff.priority←100
 [38]      :Else
 [39]          buff.priority←(⌊/_registries.priority)-10
 [40]      :EndIf
 [41]  :EndIf
 [42]  _registries,←buff
 [43]  Write ⍬
```

#.Pakkman.Client.UserSettings.ReplaceRegistry

```
  {r}←ReplaceRegistry registry
 [1]     ⍝ Replaces the definition of a Registry in the user settings.\\
 [2]     ⍝ If that Registry does not already exist an error is thrown.\\
 [3]     ⍝ Note that this function requires a URI since an alias is subject to change.\\
 [4]     ⍝ `⍵` must be an instance of the "Registry" class.\\
 [5]     ⍝ `←` is always `⍬` (shy).
 [6]   :Access Public Instance
 [7]   r←⍬
 [8]   '⍵ is not instance if the "DefineRegistry" class'Assert∨/'[DefineRegistry]'⍷⍕registry
 [9]   :If 0=≢_registries
 [10]  :OrIf ~(⊂registry.uri)∊_registries.uri
→[11]      'There is no such registry yet'Assert 0
 [12]  :EndIf
 [13]  ind←_registries.uri⍳⊂registry.uri
 [14]  _registries[ind]←registry.Get ⍬
 [15]  Write ⍬
```

#.Pakkman.Registry.ListPackages

```
  list←ListPackages uri
 [1]  ⍝ List packages.
 [2]  ⍝ Supports two special syntaxes:
 [3]  ⍝ * ListPackages '*'                ⍝ All but without versions
 [4]  ⍝ * ListPackages '*-*-*'            ⍝ All including versions (not exposed, only for internal purposes)
 [5]  ⍝ Otherwise it is treated as a RegEx
 [6]   'No registry specified'Assert 0<≢uri
 [7]   (registryPath packageID)←SplitAtLast uri
 [8]   registryPath←F.ExpandPath registryPath
 [9]   'Is not a folder'Assert F.IsDir registryPath
 [10]  list←F.ListDirs registryPath ⍝ All package IDs
 [11]  :If 0<≢list
 [12]      list←{⊃,/1↓⎕NPARTS ⍵}¨list                          ⍝ Remove path: we only need the package IDs
 [13]      :If '*.*.*'≢packageID
 [14]          list←RemoveMinorPatch¨list                     ⍝ Get rid of minor and patch number but preserve major on
 [15]          list←∪list
 [16]          :If 0<≢packageID~'*'
 [17]              :Trap 0
 [18]                  list←({0<≢packageID ⎕S 0⍠('Greedy' 0)⊣⍵}¨list)/list
 [19]              :Else
→[20]                  list←''
 [21]              :EndTrap
 [22]          :EndIf
 [23]      :EndIf
 [24]  :EndIf
 [25] ⍝Done
```

#.Pakkman.Client.UserSettings.RemoveRegistry

```
  {r}←RemoveRegistry uriOrAlias
 [1]      ⍝ Removes a Registry from the user settings.\\
 [2]      ⍝ `⍵` can be either a name or an alias.\\
 [3]      ⍝ `←` is 1 in case something was removed and 0 otherwise.
 [4]   :Access Public Instance
 [5]   :If 0=≢_registries
 [6]       r←0
 [7]   :Else
 [8]       :If 0=≢_registries
→[9]           r←0
 [10]      :Else
 [11]          :If ∨/~bool←~EqualsAliasOrName uriOrAlias
 [12]              _registries←bool/_registries
 [13]              Write ⍬
 [14]          :EndIf
 [15]          r←∨/~bool
 [16]      :EndIf
 [17]  :EndIf
```

#.Pakkman.Server.GetBestVersionNumber

```
  response←request GetBestVersionNumber(packageID path)
 [1]  ⍝ `path` is fully qualified. The packageID part might have just a major version no. or no version numer at all
 [2]   :If Reg.IsValidPackageID_WithoutVersionNo packageID
 [3]       path,←packageID,'-*'
 [4]   :ElseIf Reg.IsValidPackageID_WithMajorNo packageID
 [5]       path,←packageID,'.*'
 [6]   :EndIf
```

```
 [7]   list←⊃F.Dir path
 [8]   :If 0<≢list
 [9]       list←list[Reg.SortIndexForPackageIDs(≢path)⍋⍤1⊃list]
 [10]      path←⊃⍪¯1↑list
 [11]      ns←⎕NS''
 [12]      ns.BestVersion←(≢Reg.RemovePackageID path)⊃path
 [13]      response←request ReturnJSON Reg.JSON ns
 [14]  :Else
→[15]      response←request R.Respond 404 1
 [16]  :EndIf
 [17] ⍝Done
```

#.Pakkman.Server.OnRequest

```
  response←OnRequest request
 [1]  ⍝ Main handler for requests. Requests are first split by HTTP verb.
 [2]   :If 0=G.INI.Get'Trap:General'
 [3]       ⎕SHADOW'⎕TRAP'
 [4]       ⎕TRAP←0 'S'
 [5]   :EndIf
 [6]   :Select request.Method
 [7]   :Case 'GET'
 [8]       response←Handle_GET request
 [9]   :Case 'PUT'
 [10]      response←Handle_PUT request
 [11]  :Else
→[12]      response←ReturnNOT_IMPLEMENTED request
 [13]  :EndSelect
 [14] ⍝Done
```

#.Pakkman.Client.GetBestVersionNumber

```
  r←GetBestVersionNumber(packageID url)
 [1]   :If Reg.IsHTTP url
 [2]       client←NewClient url
 [3]       :If ':'∈RemoveHttpProtocol url
→[4]           client.Port←⊃⊃(//)⎕VFI{⍵↓⍨⍵⍳':'}RemoveHttpProtocol url
 [5]       :EndIf
 [6]       request←R.NewRequest 0
 [7]       request.URI←'/v1/packages/best_version/',packageID
 [8]       request.Headers,←⊂'Accept' 'application/json'
 [9]       res←client R.SendAndReceive request
 [10]      Assert 200≡res.StatusCode
 [11]      Assert 0<≢res.Content
 [12]      r←(⎕JSON⍠('Dialect' 'JSON5')⍎res.Content).BestVersion
 [13]  :Else
 [14]      add←Reg.GetBestVersionNumber url,packageID
 [15]      :If Reg.IsValidPackageID_WithoutVersionNo packageID
→[16]          r←packageID,add
 [17]      :Else
 [18]          r←({⍵↓⍨⍵-⍳{⍵='-'}⌽⍵}packageID),add
 [19]      :EndIf
 [20]  :EndIf
```

#.Pakkman.Client.GetPackageViaHTTP_

```
  response←{host}GetPackageViaHTTP_ packageID
 [1]  ⍝ Requests a package via HTTP(S)
 [2]  ⍝ `host` defaults to "https://localhost"
 [3]  ⍝ Port number is defined by the client settings
 [4]  ⍝ * a package ID
 [5]   host←{0<⎕NC ⍵:±⍵ ⋄ 'https://localhost'}'host'
 [6]   c←NewClient host
 [7]   :If ':'∈RemoveHttpProtocol host
→[8]       c.Port←⊃⊃(//)⎕VFI{⍵↓⍨⍵⍳':'}host
 [9]   :EndIf
 [10]  q←R.NewRequest 0
 [11]  q.URI←packageID
 [12]  response←c R.SendAndReceive q
 [13] ⍝Done
```

#.Pakkman.Client.UserSettings.GetRegistry

```
 r←GetRegistry uriOrAlias
 [1]     ⍝ Returns a Registry (as a namespace) or `⍬` in case the requested registry does not exist.
 [2]     ⍝ `⍵` can be either a name or an alias.\\
 [3]  :Access Public Instance
 [4]  'No Registry defined in the user settings'Assert 0<≢_registries
 [5]  :If 0=+/bool←EqualsAliasOrName uriOrAlias
→[6]      'uri/alias not found in user settings'Assert∨/bool
 [7]  :EndIf
 [8]  r←(bool⍳1)⊃_registries
```

#.Pakkman.Registry.Index.FindPartPackageID

```
  r←path FindPartPackageID packageName
 [1]  ⍝ `path` must point to a Pakkman Registry
```

```
 [2]  ⍝ `packageName` is the `{group}-{name}` part of a package ID
 [3]  ⍝ Returns either matrix with zero rows or a matrix with full package IDs in the first
 [4]  ⍝ column and the Registry path in the second column.
 [5]  ⍝ If the file that is expected to hold the index is not found it is created (empty).
 [6]  ⍝ See also `FindFullPackageID`
 [7]   r←0 2⍴⊂
 [8]   :If 0=##.F.Exists path,'/',GetIndexFilename
→[9]       Create path
 [10]  :EndIf
 [11]  index←(⎕UCS 10)(≠⊆⊢)1⊃'ASCII'##.F.NGET path,'/',GetIndexFilename
 [12]  :If 0<+/bool←((1+≠packageName)↑¨index)≡¨⊂packageName,'-'
 [13]      list←bool/index
 [14]      r←↑(⊂¨list),¨⊂⊂path
 [15]  :EndIf
 [16] ⍝Done
```

#.Pakkman.Client.GetDependecyTree

```
  tree←GetDependecyTree x
 [1]  ⍝ Takes `identifier` (`x`) and returns the dependencies as a matrix.
 [2]  ⍝ `identifier` must be one of:
 [3]  ⍝ * an HTTP request
 [4]  ⍝ * a folder holding a package (like file://C:\Temp\group-name-major.minor.patch\\)
 [5]  ⍝ * a path to a package in a registry (like [RegistryAlias]{packageID} or C:\MyReg\{packageID}')
 [6]  ⍝ * a package ID; Pakkmen will then attempt to find that package in the Registries defined in Client's config fil
 [7]  ⍝ Returns a dependey tree as a matrix:
 [8]  ⍝ * [;1] reflects the depth of the dependency with 1 for the root (`identifier`)
 [9]  ⍝ * [;2] PackageID of what did require that dependency
 [10] ⍝ * [;3] Is the full package ID
 [11] ⍝ * [;4] The full URL (either file://... or http(s)://...)\\
 [12]  :If initialCall←80=⎕DR x
 [13]      level←1
 [14]      identifier←x
 [15]      ⎕SHADOW'ΔTREE' ⍝ Semiglobal: ΔTREE is visible by GetDependecyTree when called recursively but not outside
 [16]      ΔTREE←CreateTree ⎕NS''
 [17]  :Else
 [18]      (level identifier)←x
 [19]  :EndIf
 [20]  identifier←ReplaceRegistryAlias identifier
 [21]  :If Reg.IsHTTP identifier
 [22]      path←{⍵↓⍨⍵⍳'/'}RemoveHttpProtocol identifier
 [23]      ΔTREE.Data⍪←level path identifier
 [24]      level GetDependecyTreeViaHTTP identifier
 [25]  :Else
 [26]      flag1←Reg.IsValidPackageID_Complete GetPackageIDFromFilename identifier
 [27]      flag2←Reg.IsValidPackageID_WithoutVersionNo GetPackageIDFromFilename identifier
 [28]      'Not a valid package ID'Assert flag1∨flag2
 [29]      :If ∧/~'/\'∊identifier  ⍝ Not a path?
 [30]          (identifier path)←ScanRegistriesFor identifier
 [31]          'Not found in any Registry'Assert 0<≠path
 [32]          identifier←path,identifier
 [33]      :EndIf
 [34]      :If Reg.IsValidPackageID_WithoutVersionNo GetPackageIDFromFilename identifier
 [35]          identifier←'file://',⊃¯1↑⊃F.Dir(RemoveFileProtocol identifier),'*'  ⍝ Get the best version
 [36]      :EndIf
 [37]      ΔTREE.Data⍪←level(GetPackageIDFromFilename identifier)identifier
 [38]      :If '.zip'≡⎕C ¯4↑identifier
 [39]          filename←RemoveFileProtocol identifier
 [40]          tempPath←Reg.GetTempDir,'/',GetPackageIDFromFilename filename
 [41]          F.RmDir tempPath
 [42]          ⎕DL 0.1
 [43]          F.MkDir tempPath
 [44]          ⎕DL 0.1
 [45]          filename DotNetZip.UnzipTo tempPath
 [46]          (level+1)GetDependecyTreeFromRegistry tempPath
 [47]          F.RmDir tempPath
 [48]      :Else
 [49]          identifier←RemoveFileProtocol identifier
 [50]          (level+1)GetDependecyTreeFromRegistry identifier
 [51]      :EndIf
 [52]      :If Reg.IsValidPackageID_Complete identifier
 [53]      :OrIf Reg.IsValidPackageID_WithoutVersionNo identifier
→[54]      ⍝ We need to scan all Registries in the client's config file
→[55]          path←ScanRegistriesFor identifier
→[56]          :If Reg.IsValidPackageID_WithoutVersionNo identifier
→[57]              identifier,←Reg.GetBestVersionNumber({0=≠⍵:⍵ ⋄ ⍵,'/'}RemoveFileProtocol path),identifier
 [58]          :EndIf
→[59]          ('Package not found in any Registry')Assert 0<≠path
→[60]          ΔTREE.Data⍪←level(GetPackageIDFromFilename identifier)(path,'/',identifier)
→[61]          (level+1)GetDependecyTreeFromRegistry path,'/',identifier
 [62]      :EndIf
 [63]  :EndIf
 [64]  :If initialCall
 [65]      :If 0<≠buildList←LoadBuildList identifier
 [66]      :AndIf 0<≠buildList←(0<≠¨buildList[;2])⌿buildList
 [67]          ΔTREE.Data⍪←(~buildList[;ΔTREE.ΔpackageID]∊ΔTREE.{Data[;ΔpackageID]}⍬)⌿buildList
```

```
 [68]      :EndIf
 [69]      tree←Prune ∆TREE
 [70]  :Else
 [71]      tree←0 4⍴⊂⍬
 [72]  :EndIf
 [73] ⍝Done
```

#.Pakkman.Registry.Index.FindFullPackageID

```
  ind←path FindFullPackageID packageName
 [1]  ⍝ `path` must point to a Pakkman Registry
 [2]  ⍝ `packageName` is simple string `{group}-{name}-{version}`
 [3]  ⍝ Returns the index of that package in the index or 0 if not found.
 [4]  ⍝ If the file that is expected to hold the index is not found it is created (empty).
 [5]  ⍝ See all `FindPartPackageID
 [6]   ind←0
 [7]   :If 0=##.F.Exists path,'/',GetIndexFilename
→[8]       Create path
 [9]   :EndIf
 [10]  index←1⊃'ASCII'##.F.NGET path,'/',GetIndexFilename
 [11]  :If 0<+/bool←(packageName,⎕UCS 10)⍷index
 [12]      ind←1+(index↑⍨bool⍳1)+.=⎕UCS 10
 [13]  :EndIf
 [14] ⍝Done
```

#.Pakkman.Server.Handle_PUT

```
  response←Handle_PUT request
 [1]   :If AcceptCredential'api-key'GetFromHeaders request.Headers   ⍝TODO⍝ Simulation: later we need to check whether
 [2]       packageID←1↓request.URI
 [3]       :If Reg.IsValidPackageID_Complete packageID
 [4]           response←SavePackage request
 [5]       :Else
→[6]           response←request R.Respond 400     ⍝ Invalid package ID and not a proper REST request, therefore 400 = B
 [7]       :EndIf
 [8]   :Else
 [9]       response←request R.Respond 401         ⍝ Unauthorized, therefore 401 = Unauthorized
 [10]  :EndIf
 [11] ⍝Done
```

#.Pakkman.Client.LoadBuildList

```
  tree←{level}LoadBuildList path
 [1]  ⍝ Reads the build list from file
 [2]  ⍝ Default for `level` is 0 (all)
 [3]  ⍝ If level is 1 then only entries with the level 1 are returned
 [4]  ⍝ If level is 2 then only entries with the level greater than are returned
 [5]  ⍝ If level is 0 all entries are returnd
 [6]   level←{0<⎕NC ⍵:±⍵ ⋄ 0}'level'
 [7]   'Invalid left argument'Assert level∊0 1 2
 [8]   filename←path,'/',Reg.BuildListFilename
 [9]   :If F.IsFile filename
 [10]      ns←0 ⎕JSON⍠('Dialect' 'JSON5')⍬1⊃F.NGET filename
 [11]      tree←⍉↑ns.{±⍵}'depth' 'packageID' 'url'
 [12]      :If level=1
→[13]          tree←(tree[;1]=1)⌿tree
 [14]      :ElseIf level=2
→[15]          tree←(tree[;1]>:1)⌿tree
 [16]      :EndIf
 [17]  :Else
 [18]      tree←0 3⍴⊂⍬
 [19]  :EndIf
 [20] ⍝Done
```

#.Pakkman.Server.FetchPackage

```
  response←request FetchPackage packageID
 [1]   :If Reg.IsValidPackageID_WithoutVersionNo packageID
→[2]       packageID,←Reg.GetBestVersionNumber G.RegistryPath,'/',packageID
 [3]   :EndIf
 [4]   :If Reg.IsValidPackageID_Complete packageID
 [5]       :If F.IsDir G.RegistryPath,'/',packageID
 [6]           response←R.NewResponse request
 [7]           response.IsFile←1
 [8]           response.FilePath←F.ExpandPath G.RegistryPath,'/',packageID,'/',packageID,'.zip'
 [9]           response.ContentType←'application/zip'
 [10]      :Else
 [11]          response←Return404 request
 [12]      :EndIf
 [13]  :Else
→[14]      response←Return404 request
 [15]  :EndIf
 [16] ⍝Done
```

#.Pakkman.Server.GetDependencies

```
  response←request GetDependencies packageID
```

```
 [1]  ⍝ Collect dependencies if not http: those need to be dealt with by the caller since this is
 [2]  ⍝ the server module, so we don't have the means to query servers at this point.
 [3]   :If Reg.IsValidPackageID_Complete packageID
 [4]       filename←(⊃¯1↑1⊃F.Dir G.RegistryPath,packageID,'*'),'/',Reg.DependenciesFilename
 [5]       :If F.IsFile filename
 [6]           data←CollectDependenciesFor filename
 [7]       :AndIf 0<≠data
 [8]           :If IsJsonRequest request.Headers
 [9]               response←request ReturnJSON Reg.JSON data
 [10]          :Else
→[11]              response←request R.Respond 401 1
 [12]          :EndIf
 [13]      :Else
 [14]          response←request R.Respond 200 0      ⍝ 200 but no content: no dependencies
 [15]      :EndIf
 [16]  :Else
→[17]      response←request R.Respond 404 1
 [18]  :EndIf
```

#.Pakkman.Client.GetAllFiles

```
  list←home GetAllFiles cfg
 [1]  ⍝ ⍵ Is an object presenting a package's config file
 [2]  ⍝ ⍺ Is the folder that hosts the package
 [3]  ⍝ ← Is a list of all cfg.source files
 [4]  ⍝ Note that if cfg.source is empty the function tries to work out what the cfg.source might be:
 [5]  ⍝ 1. If the client's config file has a default source folder defined, that is taken if it exists.
 [6]  ⍝ 2. If there is just one folder then it is assumed that this folder contains all source code files.\\
 [7]  ⍝ As a side effect this function writes back the config file in case "source" could be established.
 [8]   :If 0=≠cfg.source
 [9]       :If 0<≠default←_UserSettings.source
→[10]      :AndIf F.IsDir home,default
→[11]          list←(≠home)↓¨⊃F.Dir home,default,'/'
→[12]          ('No source code file found in ',home,default)Assert∨/(3⊃∘⎕NPARTS¨list)∈'.aplf' '.aplo' '.aplc' '.apln'
→[13]          cfg.source←default
 [14]      :Else
 [15]          list←(≠home)↓¨⊃F.Dir home,'/'
 [16]          subList←list~Reg.CFG_Name Reg.DependenciesFilename
 [17]          :If 1≠≠subList←(F.IsDir home∘,¨subList)/subList
 [18]              '"cfg.source" is undefined and cannot be established'Assert 1=≠subList
 [19]          :Else
 [20]              cfg.source←(⊃subList),'/'
 [21]              list←(≠home)↓¨⊃('recursive' 1)F.Dir home,cfg.source
 [22]          :EndIf
 [23]      :EndIf
 [24]      cfg WritePackageConfigFile home
 [25]  :Else
 [26]      :If F.IsFile home,cfg.source
 [27]          list←⊆cfg.source
 [28]      :Else
 [29]          ('"',cfg.source,'" is neither file nor folder')Assert F.IsDir home,cfg.source
 [30]          home,←(~(¯1↑home)∈'\/')/'/'
 [31]          list←(≠home)↓¨⊃('recursive' 1)F.Dir home,cfg.source,'/'
 [32]      :EndIf
 [33]  :EndIf
 [34]  list/⍨←F.IsFile projectPath∘,¨list
 [35] ⍝Done
```

#.Pakkman.Client.ScanRegistriesFor

```
  (packageID reg)←{userSettings}ScanRegistriesFor packageID
 [1]  ⍝ Scans all registries for `packageID`; the first one wins.\\
 [2]  ⍝ Note that `packageID` might be a full ({group}-{name}-{version}) or a partial one ({group}-{name} or {group}-{n
 [3]  ⍝ If it is a partial one the "best" package wins.
 [4]   reg←''
 [5]   userSettings←{0<⎕NC ⍵:⍎⍵ ⋄ _UserSettings}'userSettings'
 [6]   registries←userSettings.registries
 [7]   'No Registries defined in the Client config file'Assert 1≤≠registries
 [8]   :For uri alias :In registries.(uri alias)
 [9]       :If Reg.IsHTTP uri
 [10]          :If 0=≠list←ListPackages_ uri,packageID,'*'
 [11]              :Continue
 [12]          :Else
→[13]              ⋄⋄⋄ ⍝TODO⍝Kai
→[14]              :Leave
 [15]          :EndIf
 [16]      :ElseIf Reg.IsValidPackageID_Complete packageID
 [17]          :If F.IsDir(RemoveFileProtocol uri),'/',packageID
 [18]              reg←uri
 [19]              :Leave
 [20]          :EndIf
 [21]      :ElseIf Reg.IsValidPackageID_WithoutVersionNo packageID
 [22]          :If 0<≠list←F.ListDirs(RemoveFileProtocol uri),'/',packageID,'-*'
 [23]              packageID←⊃¯1↑{⍵[Reg.SortIndexForPackageIDs ⍵]}GetPackageIDFromFilename¨list
 [24]              reg←uri
 [25]              :Leave
```

```
 [26]          :EndIf
 [27]      :ElseIf 0<≠list←F.ListDirs(RemoveFileProtocol uri),'/',packageID
→[28]          packageID←⊃¯1↑{⍵[Reg.SortIndexForPackageIDs ⍵]}GetPackageIDFromFilename¨list
→[29]          reg←uri
→[30]          :Leave
 [31]      :EndIf
 [32]  :EndFor
 [33] ⍝Done
```

#.Pakkman.Client.GetMD5

```
 GetMD5←{
 [1]  ⍝ Takes a filename and returns an MD5 hash for that file.
 [2]      'File does not exist'Assert F.Exists ⍵:
 [3]      os←⊃⊃'.'⎕WG'APLVersion'
 [4]      os≡'M':⊃⎕SH'md5 -q ',⍵
 [5]      os≡'W':2⊃⎕SH'certutil.exe -hashfile ',⍵,' MD5'   ⍝ Tolerates slash in path
→[6]      os≡'L':{⍵↑⍨¯1+⍵⍳' '}⊃⎕SH'md5sum ',⍵
→[7]      ⍬⍬⍬
 [8]  }
```

#.Pakkman.Client.ListVersions

```
 list←ListVersions y
 [1]  uri←ReplaceRegistryAlias y
 [2]  :If Reg.IsHTTP uri
→[3]      list←ListVersions_ uri
 [4]  :Else
 [5]      list←Reg.ListVersions uri
 [6]  :EndIf
 [7] ⍝Done
```

#.Pakkman.Server.Handle_GET

```
  response←Handle_GET request
 [1]   :If (,'/')≡request.Path
→[2]       response←CreateHomePage request
 [3]   :Else
 [4]       fullPath←{⍵↓⍨'/'=⊃⍵}request.Path
 [5]       :If ∨/∨/¨'Assets/img/pakkman.png' 'Assets/img/favicon.ico'⍷¨⊂fullPath
→[6]           response←request ReturnFiles fullPath
 [7]       :ElseIf (⊂fullPath)∊'Assets/CSS/MarkAPL_screen.css' 'Assets/CSS/MarkAPL_print.css' 'Assets/CSS/screen.css' '
→[8]           response←request ReturnFiles fullPath                  ⍝ Static stuff
 [9]       :Else
 [10]          path←Reg.RemovePackageID fullPath
 [11]          packageID←(≠path)↓fullPath
 [12]          :If 0=≠path
 [13]              :If Reg.IsValidPackageID_Complete packageID
 [14]                  response←request FetchPackage packageID        ⍝ It is a fully qualified packageID
 [15]              :Else
 [16]                  response←request R.Respond 404                 ⍝ Neither a valid package ID nor a proper REST re
 [17]              :EndIf
 [18]          :ElseIf request IsREST_v1 fullPath
 [19]              response←request HandleREST_Version1 fullPath
 [20]          :Else
→[21]              response←request R.Respond 404                     ⍝ Neither a valid package ID nor a proper REST re
 [22]          :EndIf
 [23]      :EndIf
 [24]  :EndIf
 [25] ⍝Done
```

#.Pakkman.Server.ProcessCredentials

```
  G←ProcessCredentials G
 [1]  ⍝ In the long run this function will establish an array from the contents of the file "credentials.txt"
 [2]  ⍝ For the time being the data structure is just a single API key that is used to check on every
 [3]  ⍝ PUT operation.
 [4]   filename←G.RegistryPath,'/Credentials.txt'
 [5]   :If 0=F.Exists filename
→[6]       ''F.NPUT filename
 [7]   :EndIf
 [8]   G.Credentials←{0=≠⍵:⍵ ⋄ ⊆1⊃⍵}1⊃F.NGET filename 1
 [9]  ⍝Done
```

#.Pakkman.Server.ProcessTestCommands

```
  response←request ProcessTestCommands path
 [1]  ⍝ This function must only be called when G.TestFlag is true
 [2]   :If G.TestFlag
 [3]       :Select path
 [4]       :Case 'v1/GetServerPath'
 [5]           response←ReturnRegistryPath request
 [6]       :Case 'v1/RecompileIndex'
 [7]           Reg.CompileIndex G.RegistryPath
 [8]           response←request R.Respond 200
 [9]       :Case 'v1/Stop'
→[10]          1 ⎕STOP'#.Pakkman.Server.OnRequest'
```

```
→[11]           response←request R.Respond 200
 [12]       :Case 'v1/shutdown'
 [13]           ⎕TRAP←0 'S'
 [14]           response←request R.Respond 200
 [15]           ⍙SHUTDOWN←1
 [16]       :Case 'v1/crash'
→[17]           ∘∘∘                                 ⍝ Deliberate crash for testing error trapping
 [18]       :Else
→[19]           response←request R.Respond 404 1    ⍝ Unknown (new?!)
 [20]       :EndSelect
 [21]   :Else
→[22]       response←request R.Respond 401 1        ⍝ Not a valid REST version 1 request
 [23]   :EndIf
```

#.Pakkman.Client.GetPakkmanRootSpace

```
  r←GetPakkmanRootSpace targetSpace
 [1]   ⍝ `targetSpace` is where the link to a package it going to be created,
 [2]   ⍝ That rules whether we return `#` or `⎕SE` as the root.
 [3]   ⍝ As a side effect the function makes sure that the space exists.
 [4]    targetSpace←⍕targetSpace
 [5]   :If '#'=1⍴targetSpace
 [6]       r←'#._packages'
 [7]       root←#
 [8]   :ElseIf '⎕SE'≡1 ⎕C 3⍴targetSpace
→[9]       r←'⎕SE._packages'
→[10]      root←⎕SE
 [11]  :Else
→[12]      'Invalid right argument'⎕SIGNAL 11
 [13]  :EndIf
 [14]  :If 0=⎕NC r
 [15]      r ⎕NS''
 [16]  :EndIf
 [17] ⍙Done
```

#.Pakkman.Server.HandleREST_Version1

```
  response←request HandleREST_Version1 path_
 [1]   ⍝ Handles all commands defined in version 1 of the REST interface
 [2]    :If 1='/'+.=path_
 [3]       path←path_
 [4]       packageID←''
 [5]   :Else
 [6]       (path packageID)←{⍵{((⍵-1)↓⍺)(⍵↑⍺)}⍨-¯1+(⍸⍵)⍳'/'}path_
 [7]   :EndIf
 [8]   :If (⊂path)∊'v1/GetServerPath' 'v1/RecompileIndex' 'v1/Stop' 'v1/shutdown' 'v1/crash'
 [9]       response←request ProcessTestCommands path
 [10]  :Else
 [11]      path↓⍨←-'/'=¯1↑path
 [12]      :Select path
 [13]      :Case 'v1/packages'
 [14]          :If 0=≢packageID
→[15]              packageID←'*'
 [16]          :ElseIf Reg.IsValidPackageID_WithoutVersionNo packageID
→[17]          :AndIf ~⍵/'.*[]'∊⍨packageID
→[18]              packageID,←'*'
 [19]          :EndIf
 [20]          response←request GetPackageList packageID
 [21]      :Case 'v1/packages/versions'
→[22]          response←request GetVersionList packageID
 [23]      :Case 'v1/packages/details'
→[24]          response←request GetDetails packageID
 [25]      :Case 'v1/packages/dependencies'
 [26]          response←request GetDependencies packageID
 [27]      :Case 'v1/packages/best_version'
 [28]          response←request GetBestVersionNumber packageID G.RegistryPath
 [29]      :Case 'v1/groups'
→[30]          response←GetGroupList'expand'F.NormalizePath G.RegistryPath
 [31]      :Else
→[32]          ⍝ Something new?
→[33]          response←request R.Respond 400 1 ⍝ Not a valid REST version 1 request
 [34]      :EndSelect
 [35]  :EndIf
 [36] ⍙Done
```

#.Pakkman.Client.ValidatePackageFiles

```
 ValidatePackageFiles←{
 [1]       ⍺←''
 [2]       cfg←⍵
 [3]       0<cfg.⎕NC'files':cfg⊣cfg.files←,⊆⍤(0<≢∘cfg.files)⊣cfg.files
→[4]       cfg.files←''
→[5]       cfg
 [6]   }
```

#.Pakkman.Registry.Base64

```
  Base64←{
 [1]  ⍝     A←⎕A,(⎕UCS(⍳26)+⎕UCS'a'),⎕D,'+/'                        ⍝ alphabet
 [2]        ⎕IO←0
 [3]        A←'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
 [4]        ⍺←~∧/⍵∈A,'='
 [5]        ed←{⊃⍵{⍺{2⊥⍉((⌈(ρ⍵)÷⍺),⍺)⍴⍺ρ(×/⍺)↑⍵}⍵},⍉(⍵ρ2)⊤⍺⍺}/⍺}  ⍝ encode decode
 [6]        ⍺:{⍵,'='ρ⍨4|-ρ⍵}(⊂6 8 ed ⎕UCS ⍵)⌷A                      ⍝ encode strict /
→[7]        ⍝⍺:(⊂6 8 ed ⎕UCS ⍵)⌷A                                   ⍝ encode lax    x
→[8]        ⍝1:⎕UCS{x↓⍨8 6 ed A⍳⍵,'='↑⍨x←-4|-ρ⍵}⍵∩A                 ⍝ decode lax    /
→[9]        1:⎕UCS{(-+/'='=¯3↑⍵)↓⍨8 6 ed A⍳⍵}⍵∩A,'='                ⍝ decode strict x
 [10] ⍝ base64 encoding
 [11] ⍝ ⍺ 1=encode; 0=decode
 [12] ⍝ ⍵ byte string: 256∧.>:⎕ucs ⍵
 [13] ⍝ ← ⍺=1: longer (4:3) string in A-Z,a-z,0-9,+/= only
 [14] ⍝   ⍺=0: original string whence ⍵ was encoded
 [15] ⍝   whitespace in (⍵) is ignored on decode
 [16]  }
```

#.Pakkman.Registry.PROJECT_HOME

```
  r←PROJECT_HOME
 [1]   :If 0<⎕SE.⎕NC'acre'
 [2]   :AndIf 0<≠⎕SE.acre.Projects''
 [3]       r←##.AcreConfig.ProjectFolder
 [4]   :ElseIf 'CLEAR WS'≠⎕WSID
→[5]       F.PolishCurrentDir
→[6]       r←F.PWD
 [7]   :Else
→[8]       'Home directory cannot be determined'⎕SIGNAL 11
 [9]   :EndIf
```

#.Pakkman.Server.OnHouseKeeping

```
 OnHouseKeeping←{
 [1] ⍝ Purely used to detect if ΔSHUTDOWN and if so bring the program to a halt in developemnt
 [2]       0=ΔSHUTDOWN:shy←0
→[3]       _←⎕EX'ΔSHUTDOWN'
→[4]       0=##.APLTreeUtils.IsDevelopment:shy←#.Plodder.Shutdown G.INI  ⍝ Will ⎕OFF in runtime
→[5]       _←⎕NQ'#'QuitDQ
→[6]       0
 [7]  }
```

#.Pakkman.Server.GetPackageList

```
  response←request GetPackageList packageID
 [1]  ⍝ Respond to a request for some or all packages, HTML or JSON
 [2]   :If IsJsonRequest request.Headers
 [3]       :If 0=≠packageID
→[4]           data←G.RegistryPath Reg.ListPackages'*'
 [5]       :Else
 [6]           data←Reg.ListPackages G.RegistryPath,'/',packageID
 [7]       :EndIf
 [8]       response←request ReturnJSON Reg.JSON data
 [9]   :Else
→[10]      :If 0=≠packageID
→[11]      :OrIf (,'*')≡,packageID
→[12]          data←Reg.ListPackages G.RegistryPath,'/*.*.*'
 [13]      :Else
→[14]          data←Reg.ListPackages G.RegistryPath,'/',packageID
 [15]      :EndIf
→[16]      noOfVersions←{≠⍵}⌸2⊃¨'-'A.Split¨data
→[17]      IDs←Reg.Latestversions data
→[18]      data←⍪IDs
→[19]      data,←↑G.RegistryPath∘FetchDescriptionAndPackageProjectUrl¨data[;1]
→[20]      data[;1]←2⊃¨'-'A.Split¨data[;1]
→[21]      data,←noOfVersions
→[22]      IDs←1⊃¨'-'Reg.SplitAtLast¨IDs
→[23]      html←IDs PrepareHtmlPageForPackageList data
→[24]      response←request ReturnHTML html
 [25]  :EndIf
```

#.Pakkman.Client.LoadInstalledDependencies

```
  refs←targetSpace LoadInstalledDependencies path
 [1]   refs←⍬
 [2]   :If 0<≠tree←2 LoadBuildList path
→[3]       :For level requestedBy packageID url :In ↓tree
→[4]           folder←path,'/',packageID
→[5]           saveIn←GetPakkmanRootSpace targetSpace
→[6]           saveIn,←'.',packageID
→[7]           ⍬⍴''''',({⍵↓⍨⍵⍳'.'}saveIn),''' ',({⍵↑⍨⍵⍳'.'}saveIn),'⎕NS''''''
→[8]           cfg←ReadPackageConfigFile folder
→[9]           msg←⎕SE.Link.Import saveIn(folder,'/',cfg.source)
→[10]          'Link failed to import code'Assert'Imported:'{⍺≡(≠⍺)↑⍵}msg
→[11]          ref←⍬⍴saveIn
→[12]          ref.ΔURI←cfg.uri
```

```
→[13]          ref.⍙HOME←path
 [14]       :EndFor
 [15]  :EndIf
 [16] ⍝Done
```

Created by Tester2 version 2.3.0.57 from 2020-06-28