# EPOC V0.5.2 API (public methods)

## EPOCObject (virtual)

Base (virtual) EPOC class which almost all others inherit from. This includes the mechanisms for loading input data and parsing it into slots. All input data which does not match a slot is inserted into the params slot list. Its methods are available to all other class objects. This class inherits from .environment.

Slot data members:

```
        inputData       = "list"        # All data from input data file inserted here until its parsed
        dataPath        = "character"   # Path to input data file
        signature       = "Signature"   # Signature object
        epocAttributes  = "list"        # all additional parameters that do not fit into a slot
        .msglevel       = "character"   # Dictates which epocMessages are printed to stdout
        .loglevel       = "character"   # Dictates which epocMessages are written to log file
        .logfile        = "character"   # Log file name with path
        .logtrunc       = "logical"     # Is log file truncated when opened
```

Environment data members (.xData):

```
        fileConnections = "list"        # Managed file connections (externalptr(s) to EPOC
                                                                              Rcpp classes)
```

Methods:

**getAttributeNames (.Object)**
```
# Return a list of EPOC attribute names held by this object
```

**getAttribute (.Object, item=missing)**
```
# Return the objects epocAttributes list
# If  list item name is passed then return the value at that list position instead
# Parameters:
#       item    character/missing       name of list item to return (optional)
```

**setAttribute (.Object, item=missing, value)**
```
# Set the value of item in epocAttributes list.  If  item is missing and value="list" then set
# epocAttributes as passed list.
# Will append to epocAttributes list if item not already list member.
# Parameters:
#       item    character/missing       name of list item to insert value as
#       value           ANY             value to assign to the list
```

**getSlotNames (.Object)**
```
# Return a list of available slot names held by this object
```

**getSlot (.Object, item=missing)**
```
# Return value at item passed if item exists as a slot
# Parameters:
#       item    character/missing       slot from which to return data
```

**setSlot (.Object, item=missing, value)**
```
# Set value at item passed if item exists as a slot
# Parameters:
#       item            character       slot at which to assign value
#       value           ANY/            value to be assigned
```

**getFileConnection(Object, connname, filepath=missing, openmode="a")**

```
# Get the named file connection handle if it is listed else return NULL
# If listed but not open then open the connection first
# If a filepath is passed then open file if it is not listed, open or path/mode has changed
# and then store in named fileConnections list
# Return the connection or NULL if fails
# Parameters:
#       connname        character       name of connection
#       filepath        character       path to file to open (optional)
#       openmode        character       c(“a”, “w”, “r”) defaults to “a”
```

**writeFileConnection(.Object, conn)**

```
# Write a line to the specified file connection and return success
# Defaults to append mode, but will depend on the mode in which connectiion was opened.
# If a character name and filepath is passed the connection will be opened (if not open
# or path/mode differs) and the connection will be added to the fileConnections list.
# Parameters:
#       conn    externalptr/character   name of connection or pointer to connection
#       msg     ANY (hopefully)         First part of message (uses toString() to convert)
#       ...     ANY (hopefully)         Any further message parts to be pasted using sep
#       filepath        character       Path for new connection to be made if necessary,
#                                               only if conn = "character" (optional)
#       openmode        character       Open mode to be made if necessary, only if
#                                               conn = "character" (default = "a")
#       sep             character       Separator char for multipart messages (default = "")
#       eol             logical         Append an end of line character (default = TRUE)
```

**readFileConnection(.Object, conn)**

```
# Read from the specified file connection and return it
# Defaults to linenum=-1 which reads back complete file.
# If linenum=0 then next line is read back only else if linenum > 0 then that line only will be read
# back.  The next line as determined by C++ level pointer will be returned unless linenum is
# specified
# Parameters:
#       conn    externalptr/character   name of connection or pointer to connection
#       linenum         integer         Line number to read (starting at line 1
#                                               (defaults to current C++ filepointer)
```

**addFileConnection (.Object, conn="externalptr", connname="character")**

```
# Add the external pointer to file connection to the fileConnections named list using connname
# If connname already exists and is open then it will be closed and replaced.
# Parameters:
#       conn            externalptr             connection
#       connname        character               name of connection
```

**closeFileConnection(.Object, conn=missing)**

```
# Close named file connection or external pointer to connection if it is in fileConnections list.
# If conn is not specified then close all listed file connections.
# externalptr to connection still remains after being closed and can be reopened
# Returns whether connection existed and was open and therefore able to be closed
# Parameters:
#       conn    externalptr/character   name of, or pointer to connection (optional)
```

**epocMessage (.Object, msg=””, …, sep="")**
# Print verbose messages to stdout and log file as dictated by .msglevel="quiet"
# and .loglevel="quiet"
# Parameters:

| | | | |
|---|---|---|---|
| # | msg | ANY | message |
| # | … | character | more of the same |
| # | sep | character | separating character between message parts |

**epocVerboseMessage (.Object, msg=””, …, sep="")**
# Print verbose messages to stdout and log file as dictated by .msglevel="verbose" or "debug"
# and .loglevel="verbose" or "debug"
# Parameters:

| | | | |
|---|---|---|---|
| # | msg | ANY | message |
| # | … | character | more of the same |
| # | sep | character | separating character between message parts |

**epocDebugMessage (.Object, msg=””, …, sep="")**
# Print debug messages to stdout and log file as dictated by .msglevel="debug"
# and .loglevel="debug"
# Parameters:

| | | | |
|---|---|---|---|
| # | msg | character | message |
| # | … | character | more of the same |
| # | sep | character | separating character between message parts |

**epocErrorMessage (.Object, msg=””, …, sep="", halt=FALSE)**
# Print debug messages to stdout and log file.
# Call stop() after printing if halt=TRUE
# Parameters:

| | | | |
|---|---|---|---|
| # | msg | character | message |
| # | … | character | more of the same |
| # | sep | character | separating character between message parts |
| # | halt | logical | should stop() be called after printing |

**getSignature (.Object, item=missing)**
# Return the elements signature object
# If  item is passed then return the value at that slot instead
# Parameters:

| | | | |
|---|---|---|---|
| # | item | character | name of signature slot to return (optional) |

**getSignatureLine (.Object, display=FALSE)**
# Return the objects simple one-line textual representation of signature
# If display then send to stdout via epocMessage as well

**getSignatureMulti (.Object, display=FALSE)**
# Return the objects mutli-line textual representation of signature
# If display then send to stdout via epocMessage as well

## Element (virtual)

Virtual base class for all EPOC elements.  Inherits from EPOCObject.
Contains data members and functionality common to all EPOC elements. Holds spatial, timestep, state and transition information.  Its methods are available to all other derived element objects.
NOTE: Some slots have now been replaced as objects in the environment list (.xData).  These all have accessor methods.

Slot data members:

| | | |
|---|---|---|
| polygons | = "list" | # Element spatial/polygon information |
| polygonsN | = "numeric" | # Count of polygons |
| birthday | = "numeric" | # Element birthday |
| timesteps | = "list" | # Timestep information from input data file |
| timestepsN | = "numeric" | # Count of timesteps |
| recordElements | = "numeric" | # |
| currentScenarioDir | = "character" | # Directory path for current scenario |

Environment data members (.xData):

| | | |
|---|---|---|
| state | = "list" | # Current object state |
| initialState | = "list" | # Initiate object state before initialiseReplicate |
| functions | = "list" | # Function data list |
| transition | = "list" | # Transition data before state is updated |
| flags | = "list" | # Runtime flags (eg doPrint, doUpdate etc) |

Public methods:

**getBirthday (.Object)**
# Return the elements Julian day of birth in any year.

**getPolygons (.Object)**
# Return an array of spatial polygon indexes for the element.

**getState (.Object, item=missing)**
# Return the state list if it has been instantiated already
# If item passed then return value at list item in state if available
# Parameters:
#	item	character	state list member to be returned (optional)

**getTimestep (.Object, periodNum=missing)**
# Return the timestep list
# If periodNum passed then return value at index in timesteps if available
# Parameters:
#	periodNum	numeric	timestep period to be returned (optional)

**getTransition (.Object, item=missing)**
# Return the transition list
# If item passed then return value at name in transition list if available
# Parameters:
#	item	character	transition list item to be returned (optional)

**getFunctionData (.Object, item=missing)**
# Return the functions list
# If item passed then return value at name in functions list if available
# Parameters:
#	item	character	functions list item to be returned (optional)

**setState (.Object, item=missing, value)**
# Set the state list as the list value passed
# If item passed then set value at name in state list
# Parameters:

| | | | |
|---|---|---|---|
| # | item | character | state list member at which to assign value (optional) |
| # | value | ANY/list | value to be assigned |

**setTimestep (.Object, periodNum=missing, value)**
# Set the timesteps list as the value passed
# If periodNum passed then set value at index in timesteps
# Parameters:

| | | | |
|---|---|---|---|
| # | periodNum | numeric | timestep period to have value assigned (optional) |
| # | value | ANY/list | value to be assigned |

**setTransition (.Object, item=missing, value)**
# Set the transition list as the value passed
# If item passed then set value at name in transition list
# Parameters:

| | | | |
|---|---|---|---|
| # | item | character | transition list item to have value assigned (optional) |
| # | value | ANY/list | value to be assigned |

**setFunctionData (.Object, item=missing, value)**
# Set the functions list as the value passed
# If item passed then set value at name in functions list
# Parameters:

| | | | |
|---|---|---|---|
| # | item | character | functions list item to have value assigned (optional) |
| # | value | ANY/list | value to be assigned |

**sourceMethods (.Object)**
# Method loader for all Element action methods

**doFlag(.Object, flag, do=missing)**
# Check or set generic flag as specified
# Parameters:

| | | | |
|---|---|---|---|
| # | flag | character | name of flag to set |
| # | do | logical | boolean value to set flag to (optional) |

**doPrint(.Object, do=missing)**
# Check or set doPrint flag
# If a “printState” method is available for this element and doPrint==TRUE then execute
# method as last step in each period
# Parameters:

| | | | |
|---|---|---|---|
| # | do | logical | boolean value to set flag to (optional) |

**doPrintFinal(.Object, do=missing)**
# Check or set doPrintFinal flag
# If a “printState” method is available for this element and doPrintFinal==TRUE then execute
# method as last step before completing scenario simulation.
# Parameters:

| | | | |
|---|---|---|---|
| # | do | logical | boolean value to set flag to (optional) |

**doUpdate(.Object, do=missing)**
# Check or set doUpdate flag
# If a “updateState” method is available for this element and doUpdate==TRUE then execute
# method between “during” and “after” timestep/period timings
# Parameters:
#	do	logical	boolean value to set flag to (optional)

Generic methods:

These methods have empty methods bodies but may be overloaded by a specific element method. Generic methods are always called by the controller at the specified times in each period.

**initialiseReplicate (.Object, universe)**
# Called by controller prior to execution of simulation for each scenario specified
# Parameters:
#	universe	Universe	current universe object

**initialiseTransition (.Object, universe)**
# Called by controller prior to execution of simulation for each scenario specified
#	universe	Universe	current universe object

**updateState (.Object, universe)**
# Called by controller after "During" action methods and before "After" action methods
# Can be used to update element state values based upon universe state or transition data
# Parameters:
#	universe	Universe	current universe object

**printState (.Object, universe)**
# Called as an action method in element timesteps if specified
# Parameters:
#	universe	Universe	current universe object

**actionMethod (.Object, universe)**
# All timestep action methods must conform to this method template with a unique method name
# to replace ‘actionMethod’
# Parameters:
#	universe	Universe	current universe object

## Universe

Representation of the universe with its associated module elements, scenario information, spatial/polygon information and reporting formatting.
Universe input data is parsed from universe.data file found at the path passed to constructor.

Slot data members:

| | | |
|---|---|---|
| spatial | = "Spatial" | # Spatial object used to store polygon info |
| scenarios | = "list" | # List of scenario objects |
| report | = "list" | # Reporting format information |
| inputPaths | = "list" | # paths to all element class source files and input data files |
| baseDirectory | = "character" | # Current working directory |
| created | = "logical" | # Has universe already been created (as opposed to instantiated) |

Environment data members (.xData):

| | | |
|---|---|---|
| realtimeState | = "list" | # Universe state information |
| modules | = "list" | # Module list containing element lists |

Public methods:

**new("Universe", dataPath=file.path(getwd(), "data", "Universe.data.R"), msglevel=NULL, loglevel=NULL, logfile=NULL, logtrunc=NULL, create=TRUE)**

```
# Constructor for a Universe object
# Parameters:
#     dataPath      character     absolute path to universe data input file
#     msglevel      character     c("quiet", "normal", "verbose", "debug")
#     loglevel      character     c("quiet", "normal", "verbose", "debug")
#     logfile       character     file name for log file
#     logtrunc      logical       should log file be truncated before first opening
#     create        logical       should universe and its elements be built
```

**createUniverse(.Object)**

```
# Build the universe object and its module elements based on the universe.data input file
# that was passed at instantiation
```

**testInputPaths(.Object)**

```
# Test all paths to ensure that file exists at the location
# Uses Universe@inputPaths list as imported from universe data input file
# Return a vector of invalid paths
```

**getEPOCElement(.Object, modID, elemID)**

```
# Return the element in module as indicated by list index parameters
# NULL returned if element doesn't exist at indexes passed
# Parameters:
#     modID         character/numeric      module list index
#     elemID        character/numeric      element list index
```

**setEPOCElement(.Object, modID, elemID, element)**

```
# Set the element object at the parameter indexes
# Will overwrite any existing element object at that list position
# Parameters:
#     modID         character/numeric      module list index
#     elemID        character/numeric      element list index
#     element       Element                S4 Element object
```

**getRTState(.Object, item=missing)**

```
# Return the state list if it exists
# If item passed then value in state list is returned if available
# Parameters:
#       item            character       state list name to be returned (optional)
```

**getBasePath(.Object, extPath=missing)**

```
# Return the base directory path
# If extPath is passed then full path to file is returned
# Parameters:
#       extPath         character       extension to base directory (optional)
```

**getRuntimePath(.Object, extPath=missing)**

```
# Return the runtime output directory path
# If extPath is passed then full path to file is returned
# Parameters:
#       extPath         character       extension to base directory (optional)
```

**getSpatial(.Object, item=missing)**

```
# Return the spatial object if it has been instantiated already
# If item passed then return value at slot in spatial if available
# Parameters:
#       item            character       spatial slot to be returned (optional)
```

**getReport(.Object, item=missing)**

```
# Return reporting list data
# If item passed then return that list item in report list if available
# Parameters:
#       item            character       report list name to be returned (optional)
```

**getScenario(.Object, scenario=missing, item=missing)**

```
# Return scenario object if item is missing.
# Return slot in scenario if item passed
# Uses CurrentScenario from realtimeState to specify scenario if not passed
# Parameters:
#       scenario character/numeric      scenario name/number to retrieve
#       item            character       name of the scenario attribute to return
```

**getElementIndexes(.Object, moduleName=missing, element)**

```
# Returns a vector containing the module/element list indexes
#       e.g.  c(moduleListIndex, elementListIndex)
# If element="numeric" then element signature ID will be examined, not list index
# Returns an index of 0 for list names not found
# Parameters:
#       moduleName      character       list name of module (optional)
#       element         character       list name of element
```

**reSourceElementClasses(.Object)**
# Source all element class code files listed by universe

**reSourceElementMethods (.Object)**
# Source all element method code files listed by universe

**resetReporting(.Object, msglevel=NULL, loglevel=NULL, logfile=NULL, logtrunc=NULL)**
# Set up messaging, logging, debugging and heading line values
# These will be based, by order of preference as:
# values passed as parameters (eg when instantiating universe or controller)
# values set in universe data input file under report section
# lastly by a set of default values
# Parameters:

| | | | |
|---|---|---|---|
| # | msglevel | character | c("quiet", "normal", "verbose", "debug") (optional) |
| # | loglevel | character | c("quiet", "normal", "verbose", "debug") (optional) |
| # | logfile | character | name of file without path (optional) |
| # | logtrunc | logical | should logfile be truncated before opening first time |

**stopReporting(.Object)**
# Close log file if it is open and loglevel warrants it

## Scenario

Scenario class inheriting from EPOCObject. The scenario class holds data relating to a single scenario to be conducted during the simulation. Scenarios will be conducted sequentially.

Slot data members:

|  |  |  |
|---|---|---|
| scenarioNum | = "numeric" | # unique scenario number |
| yearStart | = "numeric" | # first year of scenario |
| yearEnd | = "numeric" | # final year of scenario |
| yearsN | = "numeric" | # number of years in scenario |
| firstFishingYear | = "numeric" | # first year of fishing |
| lastFishingYear | = "numeric" | # last year of fishing |
| scenarioDir | = "character" | # scenario directory |
| replicateCnt | = “numeric” | # number of replicates to perform |

Public methods:

**initialiseElementScenarios (.Object, universe="Universe")**
# For each element, initialise state files and call the elements initialiseReplicate method if available
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | universe | Universe | current universe object |

**initialiseElementTransitions (.Object, universe="Universe")**
# For each module element, call the initialiseTransition method if one exists
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | universe | Universe | current universe object |

## Signature

Class object used to store EPOC Object signature information. This class is one of few that do not inherit from EPOCObject.

Slot data members:

|  |  |  |
|---|---|---|
| ClassName | = "character" | # Name of class to instantiate this object as |
| ID | = "numeric" | # Unique numeric identifier for object |
| ID.absolute | = "numeric" | # Same as ID, for output |
| Name.full | = "character" | # Full name of object |
| Name.short | = "character" | # Abbreviated name for this object |
| Morph | = "character" |  |
| Version | = "character" | # Version number for object |
| Authors | = "character" | # Object authors |
| Last.edit | = "character" | # Date of last edit |

Public methods:

**setSignature (.Object, signatureList)**
# Set signature data member values using the named list passed as parameter
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | signatureList | list | named list of values as read from input data files |

**getSignatureItem (.Object, item)**
# Return the signature component with name = item
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | item | character | slot name from which to return value |

**getSignatureSimple(.Object)**
# Return a one-line signature for object

**displaySignature (.Object)**
# Print a message to standard out containing signature details

## Calendar

General controller for determining the calendar of events for a year. Determines the calendar of events by comparing across all parts of the ecosystem. This class is used to build and store a calendar of periods, which together determine the yearly events of a universe. Each period is bounded by a start and end day and stores information on the module elements, their actions, and timestep parameters of those actions.

Slot data members:

|  |  |  |
|---|---|---|
| periods | = "list" | # List of Period objects |
| periodsN | = "numeric" | # Number of periods in calendar |

Public methods:

**new("Calendar)**
# Standard R constructor for S4 objects.

**createCalendar(.Object, universe)**
# Create/fill this calendar object using the data loaded into the universe object
# Builds an initial list of periods with basic input period data, then inserts timestep input data
# into each period. Each elements timestep data is then converted to element periods, and
# finally an action matrix is created for each period.
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | universe | Universe | universe from which to build calendar |

**printCalendar(.Object, universe)**
# Output a textual representation of the calendar. Defaults to file but with
# toFile=FALSE will output to stdout
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | universe | Universe | Universe object to consult |
| # | toFile | logical | Print to file? (default TRUE) |

**getPeriod(.Object, periodNum)**
# Return the calendar period object specified by periodNum
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | periodNum | numeric | Number of the period to return |

**getInfoForPeriod(Object, periodNum)**
# Return a specified period information as a named list
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | periodNum | numeric | Number of the period to return info on |

**getActionMatForPeriod(Object, periodNum)**
# Return an action matrix for the period specified
# Parameters:

|  |  |  |  |
|---|---|---|---|
| # | periodNum | numeric | Number of the period to return info on |

## Period

Sub-component of the Calendar class.

Slot data members:

```
        number          = "numeric"
        day             = "numeric"             # Number of days in period
         knifeEdge      = "logical"             # Are there any knife edge timesteps
         yearPropn      = "numeric"             # Proportion of the year dedicated to period
        periodStart     = "numeric"             # Julian start day of year for period
         periodEnd      = "numeric"             # Julian end day of year for period
         modules        = "list"                # Timestep data listed by [[module]][[element]]
         periodActionMat= "list"                # The final period action matrix
```

Public methods:

**getPeriodInfo(.Object)**
# Return a list containing period slot items

**getPeriodActionMat(.Object)**
# Return the period action matrix

**getPeriodElementTSData(.Object, modnum, elemnum)**
# Return the timestep data for a particular element
# Parameters:

```
#       modnum          numeric         Module index number
#       elemnum         numeric         Element index number
```

## Controller

Main EPOC engine class used to run the simulation based on the universe passed in the constructor. It builds a calendar from the universe data which is used to structure the action matrix for the scenario simulation.

Slot data members:

| | | |
|---|---|---|
| universe | = "Universe" | # Current universe object |
| calendar | = "Calendar" | # Current calendar built from universe |
| stepthrough | = "list" | # Any simulation break parameters |

Public methods::

**new("Controller", universe, outputcalendar=TRUE, tofile=NULL, msglevel=NULL, loglevel=NULL, logfile=NULL, logtrunc=NULL, ...)**

```
# Constructor for an EPOC Controller object.
# This will automatically create a calendar from the universe data and then setup the universe.
# All reporting parameters override existing universe constructor arguments or universe input data
# values.
# Parameters:
#       universe          Universe        universe object upon which to act
#       outputcalendar    logical         should calendar be output, overrides universe input
#       tofile            logical         should it go to file rather than screen, overrides
                                                               universe input data
#       msglevel          character       c("quiet", "normal", "verbose", "debug")
#       loglevel          character       c("quiet", "normal", "verbose", "debug")
#       logfile           character       file name for log file
#       logtrunc          logical         should log file be truncated before first opening
```

**outputCalendar(.Object, tofile=NULL)**

```
# Display the calendar associated with this universe to file by default
# If toFile=FALSE then the calendar will be output to screen
# Parameter:
#       tofile            logical   Should the calendar be output to screen or file(default)
```

**runSimulation (.Object, epocdebug=missing, timer=FALSE, forceGC=FALSE)**

```
# Wrapper to run the simulation using the universe passed at instantiation, and show timing
# Parameters:
#       epocdebug         character       break simulation at the end of one of one or more of
#                                         c("pre_action", "post_action", "period", "year",
#                                         "scenario") or the break at a particular value
#                                         e.g. c(action="migrate", year=1950)
#       timer             logical         should timing be printed to screen (default FALSE)
#       forceGC           logical         should a garbage collection event be forced before
#                                         running the simulation
```

**getCalendar (.Object)**

# Return the calendar object held by the controller

**getUniverse (.Object)**

# Return the universe object held by the controller

**closeDataFiles (.Object)**

# Close all open data files held by each element
# Each Element opens its own data file during initialisation. Controller uses this method.

## Support

This file contains a number of support methods which are not related to EPOC classes but provide support functionality during the course of their business.

Public methods:

**dayFromDate (day="numeric", month="numeric")**
```
# Convert from day and month integers to a single day in year integer
# Parameters:
#       day             numeric         day in month
#       month           numeric         month in year
```

**fixedFieldLength (value, width="integer", sig="integer", dec="integer")**
```
# Return a fixed field character value
# Parameters:
#       value   numeric/logical/character       value
#       width           integer         desired length of string
#       sig             integer         number of decimal places
#       dec             integer         if NULL then unlimited otherwise limit to number
```

**getNoCase (x = “list”, element = “character”)**
```
# Return the list element from the named list based on a non-case sensitive name
# Returns NULL if not in list
# Parameters:
#       x               list            named list
#       element         character       case insensitive name
```

**asCSVCharacter(values, ..., sep=" ")**
```
# Return a character vector with values pass separated by sep
# Parameters:
#       values  vector/list/matrix      input values to be concatenated
#       ...     vector/list/matrix      more values
#       sep     character               character separator (default = ",")
```

**fromCSVCharacter(values, type="character", sep=",")**
```
# Return a character(default) vector from values passed split by sep
# Treats a list as separate items to be dealt with separately and a list returned
# Parameters:
#       values  character/list          input values to be split
#       type    character               type of vector to return (default = "character")
#       sep     character               character separator (default = ",")
```

**setCPPMethod(method, element, body, ...)**
```
# Provides an EPOC specific alternative to the inline package's setCMethod() function.
# Compiles body C++ code using R's packaging system, rTools and the 'inline' package.
# Sets a generic method (if needed) and an EPOC Action method for class element
# See it's documentation for details of parameters.
```

**setEPOCLibMethod(method, element, libpath, libfn=method)**
```
# Used to create an EPOC Action wrapper method around a .Call() to libfn in a shared library
# at libpath.  Sets a generic (if needed) and an S4 method for named class element
```