A Graphical User Interface Builder for Data Entry and Regression Model Prediction using Windows/NT S-PLUS

Frank E Harrell Jr
Division of Biostatistics and Epidemiology
Department of Health Evaluation Sciences
University of Virginia School of Medicine
Box 600 Charlottesville VA 22908 USA
fharrell@virginia.edu
hesweb1.med.virginia.edu/biostat

1999 S-PLUS INTERNATIONAL USER CONFERENCE
NEW ORLEANS, LA
20 OCTOBER 1999

Outline

1. The Design library
2. Function generators
3. Purpose of the `Dialog.data.frame` function
4. Purpose of `Dialog.Design`
5. Basic Example: `Dialog.data.frame`
6. Advanced Example: Display predictions and C.L. from Cox, logistic, and OLS models — `Dialog.Design`

The Design Library

- For Unix and Windows/NT S-PLUS
- Facilitates advanced modeling, presentation graphics, and validation
- Makes fit objects preserve information for automatic
  - hypothesis tests (e.g., linearity of effects)
  - plotting of fitted functions
  - presentation graphics (e.g., nomograms)
  - "safe" predictions (remembers transformation parameters)
  - typesetting of algebraic form of fitted model
  - generation of S-PLUS functions to obtain quick predictions

Function Generators

- Significant savings of coding by analyst
- Generators in Design:
  - `Function`: computes $X\hat{\beta}$
  - `Survival`: computes $\hat{S}(t|X\beta)$ for survival models
  - `Quantile`: computes $\hat{S}^{-1}(p|X\beta)$ for survival models
  - `Mean`: computes mean survival time given $X\beta$
  - `Hazard`: computes hazard function for parametric survival models
  - `Dialog`: generates GUI calls and callback function
  - Planned: `StatServe`: generate analytic, HTML, and possibly Java code to install application on StatServer

Methods of Programming Function Generators

```
Function ← function(obj) {
 str ← paste('function(a,b)', stuff,'')
 # stuff can be some complex derivation from obj
 eval(parse(text=str))
}

Function ← function(obj) {
 g ← function(a,b) {
   ... generic code ...
   }
 # Make function to be self-contained
 g$a ← obj$z
 g$b ← ....
 # These make it as if the user invokes g with
 # g(a=obj$z, b=....)
 g
}
```


`Quantile` Example using `cph` Fit Object

```
survq <- Quantile(survivalfit)
# Generates the following function
function(q = 0.5, lp = 0, stratum = 1,
type = c("step", "polygon"),
time = c(0, 3, 4, 5, 6,
. . . .
  1495, 1524, 1546, 1607, 1687,
  1699, 1776, 2029),
surv =structure(.Data = c(1,
. . . .
  0.989189, 0.967955, 0.953179, 0.939128, .
  0.263251, 0.2595, 0.255594, 0.25141, . . .
0.234181),
type = "efron"))
{
  type <- match.arg(type)
  if(length(stratum) > 1)
    stop("does not handle vector stratum")
  if(is.list(time)) {
    time <- time[[stratum]]
```

```
    surv <- surv[[stratum]]
  }
  Q <- matrix(NA, nrow = length(lp),
        ncol = length(q),
        dimnames = list(names(lp),
    format(q)))
  for(j in 1:length(lp)) {
    s <- surv^exp(lp[j])
    if(type == "polygon")
      Q[j,  ] <- approx(s, time, q)$y
    else for(i in 1:length(q))
        Q[j, i] <- min(time[s <= q[i]])
        ##is NA if none
  }
  drop(Q)
}
```


Dialog.data.frame

- Takes a user data frame and creates a GUI to input one observation like that
- Can optionally add range checks so that range of entered data is in range of original data frame (or user can specify limits)
- Instead of a data frame, user can specify all data attributes
- User specifies a list of functions to compute on each observation. Values of these are displayed in read–only fields of the GUI
- Generates
  - menu function: user input $\longrightarrow$ data frame
  - runmenu function: so user can easily issue `guiDisplayDialog` command
  - callback function (real workhorse)
  - many GUI calls (especially `guiCreate`)

`Dialog.Design`

- Purpose is to get predictions from model fits without having to re–program GUI when models are updated
- User specifies a series of “packaged” fit objects using Design’s `fitPar` function
- Extracts data attributes from model fits to hand to `Dialog.data.frame`
- Compiles list of all predictor variables used in any model
- User can specify that entered values for predictor variables are to be in limits of data used to fit the models
- All functions to give to `Dialog.data.frame` are derived from `predict` (actually `predict.Design`)
- Can add `predict.dataRep` to the list if the

user has used the Hmisc library’s `dataRep` (“data representativeness”) function to store information about frequency of occurence of types of subjects used to build the models

- User can specify that confidence limits (individual, mean, or both) are to be put beside predicted values
- Can add a function provided by `Dialog` to the list to plot all results

`Dialog.data.frame` Example

- Create a data frame
- Generate a GUI to input data like that
- Also display $\sqrt{}$ of one of the variables plus the value of a binary variable

```
test ← expand.grid(age=21:50,
                   sex=c('female','male'),
                   sick=0:1)
Dialog(test,
  fun=function(z)
       round(sqrt(z$age)+z$sick,1),
  funlabel='Square root of age, +sick',
  limits=list(age=c(10,70)),
  fungroups=c('Calculated Values'=1))
runmenu.test()
```


Generated CallBack Function

```
function(df, NAMES = c("age", "sex", "sick"),
  TYPE = c("integer", "factor", "binary"),
  LIMITS = list(
  age = c(10, 70)), FUN = function(z)
  round(sqrt(z$age) + z$sick, 1),
  PREFIX = "test.",
  FUNARGTYPE = "list", FUNEXTRA = list(), AUXFUN
   = list(), AUXEXTRA = list())
{
  prop <- cbGetActiveProp(df)
  val <- cbGetCurrValue(df, prop)
  m <- length(NAMES)
  k <- (pnams <- paste(PREFIX, NAMES, sep = "")) %in%
    prop
  if(any(k)) {
    nam <- NAMES[k]
    pnam <- pnams[k]
    if(TYPE[k] %in% c("integer", "float"))
      {
      if(val != "" && !all.is.numeric(
        val)) {
        df <- cbSetCurrValue(df,
          pnam, "")
        guiCreate("MessageBox",
          String =
          "The value entered is not a legal number."
          )
      }
      val <- as.numeric(val)
      if(!is.na(val) && length(LIMITS
        ) && nam %in% names(
        LIMITS)) {
        r <- LIMITS[[nam]]
        if(val < r[1] || val >
          r[2]) {
```

```
          df <- cbSetCurrValue(
          df, pnam, "")
          fr <- format(r)
          guiCreate(
          "MessageBox",
          String = paste(
          "Value of ", nam,
          " is required to be in the range [",
          fr[1], ",", fr[2],
          "].", sep = ""))
        }
      }
    }
  }
  if(IsInitDialogMessage(df))
    return(df)
  else if(cbIsOkMessage(df)) {
  }
  else if(cbIsCancelMessage(df)) {
  }
  else if(cbIsApplyMessage(df)) {
# Am I called when the Apply buttom is pushed?
    if(length(FUN)) {
      vals <- df[2:(m + 1), "value"]
      d <- vector("list", m)
      names(d) <- NAMES
      for(j in 1:m)
        d[[j]] <- if(TYPE[j] %in%
          c("string", "factor")
          ) vals[j] else if(
          TYPE[j] == "binary")
          1 * as.logical(vals[j
          ])
        else if(TYPE[j] ==
          "logical")
          as.logical(vals[j])
        else as.numeric(vals[j]
          )
```



```
      if(FUNARGTYPE == "data.frame")
        d <- as.data.frame(d)
      fu <- if(length(FUNEXTRA)) FUN(
          d, FUNEXTRA) else FUN(
          d)
      for(i in 1:length(fu))
        df <- cbSetCurrValue(df,
          paste(PREFIX, "FUN",
          i, sep = ""), fu[[i]]
          )
      if(length(AUXFUN)) {
        if(is.character(AUXFUN)
          )
          AUXFUN <- eval(
          as.name(AUXFUN),
          local = F)
        results <- attr(fu,
          "results")
        if(!length(results))
          warning(
          "auxFun given but result of fun did not have a \"results\" attr
          )
        else {
          if(length(AUXEXTRA))
          AUXFUN(results,
            AUXEXTRA)
          else AUXFUN(results)
        }
      }
    }
  }
  df
}
```

`Dialog.Design` Example

- From online help for `Dialog`
- Uses the `support` data frame from our Web page
- Develop a least squares model for log hospital costs, binary logistic model for probability of death in hospital, Cox model for survival time

```
costfit ← ols(log(totcst) ∼ dzgroup +
              rcs(age,4) + race + sex +
              rcs(meanbp,5) + pol(scoma,2),
              data=support,
              subset=totcst>0)
hospmortfit ← lrm(hospdead ∼ rcs(age,4) +
                  sex +  rcs(meanbp,5) +
                  rcs(crea,4),
                  data=support)
survivalfit ← cph(Surv(d.time, death) ∼
                  dzgroup*rcs(age,4),
                  surv=T, data=support)
```



```
drep ← dataRep(∼ dzgroup +
               roundN(age,20),
               data=support)

surv ← Survival(survivalfit)
surv1 ← function(lp) surv(365, lp)
surv2 ← function(lp) surv(365*2, lp)
survq ← Quantile(survivalfit)
survmed ← function(lp) survq(.5, lp)

Dialog(fitPar('costfit', lp=F,
              conf.type='both',
       fun=list('Median Cost ($/1000)'=
                function(y)exp(y)/1000),
                     fun.round=0),
      fitPar('hospmortfit', lp=F,
             conf.type='mean',
       fun=list(
  'Probability of Death in Hospital'=
               plogis),label=''),
      fitPar('survivalfit', lp=F,
```

```
            conf.type='none',
        fun=list('1-Year Survival'=surv1,
                 '2-Year Survival'=surv2,
  'Median Survival Time [days]'=survmed),
            fun.round=c(2,2,0),
            label='Survival Predictions'),
      vary=list(race=levels(support$race)),
      basename='SUPPORT',
      header='SUPPORT Predictive Models',
      prompts=c(scoma='SUPPORT Coma Score',
                sex='Sex'),
      defaultAuto='binfactor',
      limits='data', datarep=drep)
runmenu.SUPPORT()
```


Generated CallBack Function

```
function(df, NAMES = c("dzgroup", "age", "sex", "meanbp",
  "scoma", "crea"), TYPE = c("factor", "float", "factor",
  "float", "float", "float"), LIMITS = list(dzgroup =
  structure(.Data = c(1, 2), .Label = c(
  "ARF/MOSF w/Sepsis", "MOSF w/Malig"), class = "factor"),
  age = structure(.Data = c(18.0419921875,
  101.847961425781), class = "labelled"), race = structure(
  .Data = c(1, 2), .Label = c("white", "hispanic"), class
   = "factor"), sex = structure(.Data = c(1, 2), .Label =
  c("female", "male"), class = "factor"), meanbp =
  structure(.Data = c(0, 180), class = "labelled"), scoma
   = structure(.Data = c(0, 100), class = "labelled"),
  crea = structure(.Data = c(0.29998779296875,
  11.798828125), class = "labelled")), FUN = function(df,
  FITINFO = list(list(fitname = "costfit", label =
  "Median Cost ($/1000)", lp = F, lplabel =
  "Median Cost ($/1000)", lp.round = 2, conf.int = 0.95,
  conf.type = "both", fun = list("Median Cost ($/1000)" =
  function(y)
exp(y)/1000), fun.round = 0, olsFit = T, varnames = c("dzgroup",
  "age", "race", "sex", "meanbp", "scoma")), list(fitname
   = "hospmortfit", label = "", lp = F, lplabel = "",
  lp.round = 2, conf.int = 0.95, conf.type = "mean", fun
   = list("Probability of Death in Hospital" = function(q,
  location = 0, scale = 1)
{
  p <- .Internal(plogis(q, par1 = location, par2 = scale),
    "S_pfuns", T, 8)
  if(!is.null(Names <- names(q)))
    names(p) <- rep(Names, length = length(p))
  p
}
), fun.round = 2, olsFit = F, varnames = c("age", "sex",
  "meanbp", "crea")), list(fitname = "survivalfit", label
   = "Survival Predictions", lp = F, lplabel =
```

```
  "Survival Predictions", lp.round = 2, conf.int = 0.95,
  conf.type = "none", fun = list("1-Year Survival" =
  function(lp)
surv(365, lp), "2-Year Survival" = function(lp)
surv(365 * 2, lp), "Median Survival Time [days]" = function(lp)
survq(0.5, lp)), fun.round = c(2, 2, 0), olsFit = F, varnames =
  c("dzgroup", "age", "dzgroup * age"))), vary = list(race
   = c("white", "black", "asian", "other", "hispanic")),
  datarep = structure(.Data = list(call = dataRep(formula
   =  ~ dzgroup + roundN(age, 20), data = support),
  formula = structure(.Data =  ~ dzgroup + roundN(age, 20),
  class = "formula"), n = 1000, names = c("dzgroup", "age"
  ), types = structure(.Data = c("exact categorical",
  "round"), .Names = c("dzgroup", "age")), parms =
  structure(.Data = c(
  "ARF/MOSF w/Sepsis COPD CHF Cirrhosis Coma Colon Cancer L
ung Cancer MOSF w/Malig",
  "to nearest 20"), .Names = c("dzgroup", "age")),
  margfreq = list(dzgroup = structure(.Data = c(391, 116,
  143, 55, 60, 49, 100, 86), .Dim = 8, .Names = c(
  "ARF/MOSF w/Sepsis", "COPD", "CHF", "Cirrhosis", "Coma",
  "Colon Cancer", "Lung Cancer", "MOSF w/Malig"),
  .Dimnames = list(.Names = c("ARF/MOSF w/Sepsis", "COPD",
  "CHF", "Cirrhosis", "Coma", "Colon Cancer",
  "Lung Cancer", "MOSF w/Malig"))), age = structure(.Data
   = c(35, 191, 420, 335, 19), .Dim = 5, .Names = c("20",
  "40", "60", "80", "100"), .Dimnames = list(.Names = c(
  "20", "40", "60", "80", "100")))), percentiles = list(
  dzgroup = NULL, age = structure(.Data = c(18.0419921875,
  22.1978846740723, 26.0216348266602, 29.0773460388184,
    . . . .
  101.847961425781), .Names = c("  0%", "  1%", "  2%",
  "  3%", "  4%", "  5%", "  6%", "  7%", "  8%", "  9%",
    . . . .
  " 94%", " 95%", " 96%", " 97%", " 98%", " 99%", "100%"))
  ), X = structure(.Data = list(dzgroup = structure(.Data
   = c(1, 2, 3, 5, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4,
  5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 5), .Label
```



```
   = c("ARF/MOSF w/Sepsis", "COPD", "CHF", "Cirrhosis",
  "Coma", "Colon Cancer", "Lung Cancer", "MOSF w/Malig"),
  class = "factor"), age = structure(.Data = c(20, 20, 20,
  20, 20, 40, 40, 40, 40, 40, 40, 40, 40, 60, 60, 60, 60,
  60, 60, 60, 60, 80, 80, 80, 80, 80, 80, 80, 80, 100, 100,
  100, 100), class = "roundN", label = "Age", name = "age",
  percentiles = structure(.Data = c(18.0419921875,
    . . . .
  101.847961425781), .Names = c("  0%", "  1%", "  2%",
    . . . .
  " 94%", " 95%", " 96%", " 97%", " 98%", " 99%", "100%")),
  tolerance = 20)), out.attrs = list(dim = structure(.Data
   = c(8, 5), .Names = c("dzgroup", "age")), dimnames =
  list(dzgroup = c("dzgroup=ARF/MOSF w/Sepsis",
  "dzgroup=COPD", "dzgroup=CHF", "dzgroup=Cirrhosis",
  "dzgroup=Coma", "dzgroup=Colon Cancer",
  "dzgroup=Lung Cancer", "dzgroup=MOSF w/Malig"), age = c(
  "age= 20", "age= 40", "age= 60", "age= 80", "age=100"))),
  row.names = c("1", "2", "3", "5", "8", "9", "10", "11",
  "12", "13", "14", "15", "16", "17", "18", "19", "20",
  "21", "22", "23", "24", "25", "26", "27", "28", "29",
  "30", "31", "32", "33", "34", "35", "37"), class =
  "data.frame"), count = c(23, 2, 1, 3, 6, 83, 4, 12, 26,
  13, 11, 18, 24, 142, 48, 62, 28, 20, 27, 60, 33, 138, 57,
  62, 1, 21, 11, 22, 23, 5, 5, 6, 3), na.action = NULL),
  class = "dataRep"))
{
  f <- function(p, g = function(x)
  x, conf.type, conf.int, dig)
  {
    s <- g(c(p$linear.predictors, p$lower, p$upper,
      p$ilower, p$iupper))
    s <- format(round(s, dig), nsmall = dig)
    n <- length(p$linear.predictors)
    yhat <- s[1:n]
    if(conf.type != "none") {
      lower <- s[(n + 1):(2 * n)]
      upper <- s[(2 * n + 1):(3 * n)]
```

```
  }
  if(conf.type == "both") {
    ilower <- s[(3 * n + 1):(4 * n)]
    iupper <- s[(4 * n + 1):(5 * n)]
  }
  switch(conf.type,
    none = yhat,
    mean = ,
    individual = paste(yhat, "  ", format(
      100 * conf.int), "% CL[", lower,
      ", ", upper, "]", sep = ""),
    both = paste(yhat, "  ", format(100 *
      conf.int), "% CLm[", lower, ", ",
      upper, "] CLi[", ilower, ", ",
      iupper, "]", sep = ""))
}
Nvary <- if(length(vary)) length(vary[[1]]) else 1
if(length(vary)) {
  attr(df, "class") <- NULL
# make into ordinary list
  df[[names(vary)]] <- vary[[1]]
  df <- expand.grid(df)
}
RES <- NULL
for(j in 1:length(FITINFO)) {
  fitinfo <- FITINFO[[j]]
  FITOBJ <- eval(as.name(fitinfo[[1]]), local = F)
  nvary <- if(Nvary > 1 && names(vary) %nin%
    fitinfo$varnames) 1 else Nvary
  morefun <- fitinfo$fun
  lp <- fitinfo$lp
  conf.int <- fitinfo$conf.int
  conf.type <- fitinfo$conf.type
  lp.round <- fitinfo$lp.round
  fun.round <- fitinfo$fun.round
  lmf <- length(morefun)
  dfj <- if(nvary == 1 && Nvary > 1) df[1,  ]
     else df
```



```
p <- switch(conf.type,
  none = list(linear.predictors = predict(
    FITOBJ, dfj)),
  mean = predict(FITOBJ, dfj, conf.int =
    conf.int, conf.type = "mean"),
  individual = predict(FITOBJ, dfj,
    conf.int = conf.int, conf.type
     = "individual"),
  both = {
    a <- predict(FITOBJ, dfj,
      conf.int = conf.int, conf.type
       = "mean")
    b <- predict(FITOBJ, dfj,
      conf.int = conf.int, conf.type
       = "individual")
    a$ilower <- b$lower
    a$iupper <- b$upper
    a
  }
  )
fitinfo$estimates <- p
FITINFO[[j]] <- fitinfo
start <- 0
n <- length(p$linear.predictors)
res <- character(nvary * lp + nvary * lmf)
if(lp) {
  res[(start + 1):(start + n)] <- f(p,
    conf.type = conf.type, conf.int
     = conf.int, dig = lp.round)
  start <- start + n
}
if(lmf)
  for(i in 1:lmf) {
    res[(start + 1):(start + n)] <-
      f(p, morefun[[i]], conf.type,
      conf.int, dig = fun.round[i])
    start <- start + n
  }
```

```
    RES <- c(RES, res)
  }
  FITINFO$data <- df
  FITINFO$vary <- vary
  if(length(datarep)) {
    p <- predict(datarep, df[1,  ])
    p$vars <- names(datarep$X)
    FITINFO$datarep <- p
    RES <- c(RES, paste(ncol(datarep$X),
      " variables:", p$count,
      "  Single variable:", p$worst.margfreq,
      sep = ""))
  }
  attr(RES, "results") <- FITINFO
  RES
}
, PREFIX = "SUPPORT.", FUNARGTYPE = "data.frame", FUNEXTRA =
  list(), AUXFUN = "plotDialogResults", AUXEXTRA = list()
  )
{
  prop <- cbGetActiveProp(df)
  val <- cbGetCurrValue(df, prop)
  m <- length(NAMES)
  k <- (pnams <- paste(PREFIX, NAMES, sep = "")) %in% prop
  if(any(k)) {
    nam <- NAMES[k]
    pnam <- pnams[k]
    if(TYPE[k] %in% c("integer", "float")) {
      if(val != "" && !all.is.numeric(val)) {
        df <- cbSetCurrValue(df, pnam,
          "")
        guiCreate("MessageBox", String
           =
          "The value entered is not a leg
al number."
          )
      }
      val <- as.numeric(val)
```



```
      if(!is.na(val) && length(LIMITS) && nam %
in%
        names(LIMITS)) {
        r <- LIMITS[[nam]]
        if(val < r[1] || val > r[2]) {
          df <- cbSetCurrValue(df, pnam,
            "")
          fr <- format(r)
          guiCreate("MessageBox", String
             = paste("Value of ", nam,
            " is required to be in the ra
nge [",
            fr[1], ",", fr[2], "].", sep
             = ""))
        }
      }
    }
  }
  if(IsInitDialogMessage(df))
    return(df)
  else if(cbIsOkMessage(df)) {
  }
  else if(cbIsCancelMessage(df)) {
  }
  else if(cbIsApplyMessage(df)) {
# Am I called when the Apply buttom is pushed?
    if(length(FUN)) {
      vals <- df[2:(m + 1), "value"]
      d <- vector("list", m)
      names(d) <- NAMES
      for(j in 1:m)
        d[[j]] <- if(TYPE[j] %in% c(
          "string", "factor")) vals[j]
           else if(TYPE[j] == "binary")
          1 * as.logical(vals[j])
        else if(TYPE[j] == "logical")
          as.logical(vals[j])
```

```
        else as.numeric(vals[j])
      if(FUNARGTYPE == "data.frame")
        d <- as.data.frame(d)
      fu <- if(length(FUNEXTRA)) FUN(d,
          FUNEXTRA) else FUN(d)
      for(i in 1:length(fu))
        df <- cbSetCurrValue(df, paste(
          PREFIX, "FUN", i, sep = ""),
          fu[[i]])
      if(length(AUXFUN)) {
        if(is.character(AUXFUN))
          AUXFUN <- eval(as.name(AUXFUN),
            local = F)
        results <- attr(fu, "results")
        if(!length(results))
          warning(
            "auxFun given but result of f
un did not have a \"results\" attribute"
            )
        else {
          if(length(AUXEXTRA))
            AUXFUN(results, AUXEXTRA)
          else AUXFUN(results)
        }
      }
    }
  }
  df
}
```


Acknowledgements

Development of the `Dialog` function was funded by Roche Pharmaceuticals for a project directed by Chris Barker, with whom interactions resulted in many new ideas as well as improvements in the software.

Abstract

This talk describes a GUI generator that can save many of the tedious steps in creating a GUI application in which the user enters variable values and obtains computed results. An example will be given of the use of a high–level function that takes as input a series of regression model fit objects and automatically creates a GUI menu from them. This menu is used to obtain user input of all of the independent variables and to report predicted values and confidence intervals for all of the model fits. Different models are allowed to have different independent variables.

This approach was used to build a risk prediction instrument to depict the benefits of a pharmaceutical company's new drug. In addition to showing predicted values, the GUI displays the number of patients in the randomized clinical trial database that were similar to the new patient for whom predictions are requested, and it displays results graphically.