```
/* ---------------------------------------------------------------------- */
/* --                SQL Interface Class for mySQL Access                -- */
/* --                                                                    -- */
/* --                Thomas Jungmann thj@gmx.net                         -- */
/* --                last modified 03.12.2000                            -- */
/* --                This interface uses RexxSQL by Mark Hessling        -- */
/* ---------------------------------------------------------------------- */

.local~sessionTimeOut = 20                      --Minutes after a session is closed after last user action
.local~DBConnectionIdentifier ="c1"             --Currently an arbitrary string
.local~DBUserName = "calendar"                  --MySQL-Username
.local~DBPassword = "wauwau"                    --MySQL-Password for User
.local~DBDatabase = "calendarstore"             --MySQL-Database name
.local~DBHost = "matush.wi-inf.uni-essen.de"    --FQ Hostname of MySQL-Server


::CLASS SQLInterface PUBLIC

    ::METHOD INIT
        --Constructor--------------
        /* loads the required library 'RexxSQL' automagically   */
        /* and initiates the connection to the database         */
        ---------------------------
        if rxFuncQuery("SQLLoadFuncs") then do
            call rxFuncAdd "SQLLoadFuncs","rexxsql","SQLLoadFuncs"
            call SQLLoadFuncs
        end
        self~establishDBConnection()

    ::METHOD UNINIT
        --Destructor---------------
        /* Disconnects from database and                */
        /* unloads the library 'RexxSQL'                */
        ---------------------------
        self~dropDBConnection()
        call SQLDropFuncs "UNLOAD"

```

```
39      ::METHOD establishDBConnection PRIVATE
40          ----------------------------
41          /* need to connect to database before querys can be issued */
42          ----------------------------
43          res = SQLConnect(.DBConnectionIdentifier,.DBUserName,.DBPassword,.DBDatabase,.DBHost)
44
45      ::METHOD dropDBConnection
46          ----------------------------
47          /* disconnects from database */
48          ----------------------------
49          res = SQLDisconnect(.DBConnectionIdentifier)
50
51      ::METHOD getAllUsers PRIVATE
52          ----------------------------
53          /*  returns a Rexx-Stem Variable (Array) with all users.
54              Can be accessed from the calling script like this:
55              a. = sqli~getAllUsers()
56              do i=1 to a.0
57                  say a.uID.i a.uniqueName.i a.password.i
58              end                                          */
59          ----------------------------
60          res = SQLCommand(SQLout,"select * from user;")
61          return SQLout.
62
63      ::METHOD getNewUID PRIVATE
64          ----------------------------
65          /*  internal Method, used for finding a new user-ID
66              (like MySQLs auto_increment - function)     */
67          ----------------------------
68          cmd = "select MAX(uID) as mxid from user;"
69          res = SQLCommand(SQLout,cmd)
70          maxID = SQLout.mxid.1
71          return (maxID+1)
72
73      ::METHOD createNewUser
74          ----------------------------
75          /*  Inserts a new User into the database
76              Arguments:  uName       = Username
```

```
77                          uPassword   = Users password
78              Returns:        -1      on Error while DB-Access
79                              -2      bad username or password
80                          UserID      when successful */
81          ---------------------------
82          parse arg uName,uPasswd
83          if uName="-1" | uPasswd="-1" then       /* "-1" is reserved as a flag value and should not  */
84              return -2                           /* be used as UserName/Password                     */
85          uid = self~getNewUID()
86          cmd = "insert into user values("||uid||","||d2c(39)||uName||d2c(39)||","||d2c(39)||uPasswd||d2c(39)||");"
87          res = SQLCommand(SQLout,cmd)
88          if res \= 0 then
89              return -1
90          return(uid)
91
92      ::METHOD deleteUser
93          ---------------------------
94          /*  Permanently deletes User from database
95              Arguments:  uID         = UserID
96              Returns:        0       success
97                          other       failure */
98          ---------------------------
99          parse arg uid
100         cmd = "delete from user where uid="||uid||";"
101         res = SQLCommand(SQLout,cmd)
102         return res                      /* res=0 => success, else failure */
103
104     ::METHOD getUserID
105         ---------------------------
106         /*  Retrieves the UserID for a given Nickname, so Users don't have
107             to remember IDs but only Names.
108             Arguments:  uniqueName  = Users Nickname
109             Returns:    -1          on DB-Access Error
110                         -2          User not found
111                         UserID      when Nickname was found         */
112         ---------------------------
113         parse arg uniqueName
114         cmd = "select uID from user where uniqueName="||d2c(39)||uniqueName||d2c(39)||";"
```

```
115         res = SQLCommand(SQLout,cmd)
116         if res \= 0 then
117             return -1                   /* Error while DB-Access */
118         if SQLout.uID.0 then            /* SQLout.uID.0 = Resultcount of SQLCommand: 1=TRUE=found */
119             return SQLout.uID.1         /* SQLout.uID.1 contains the userID         */
120         else
121             return -2                   /* SQLCommand returned empty list => Name not found! */
122
123     ::METHOD getUserName
124         ---------------------------
125         /*  Retrieves the UserName (Nickname) for a given UserID
126             Arguments:  uID         = UserID
127             Returns:    -1          on Error
128                         userName    when uID exists                 */
129         ---------------------------
130         parse arg uID
131         cmd = "select uniqueName from user where uID="||d2c(39)||uID||d2c(39)||";"
132         res = SQLCommand(SQLout,cmd)
133         if SQLout.uniqueName.0 then     /* SQLout.uniqueName.0 = Resultcount of SQLCommand: 1=TRUE=found! */
134             return SQLout.uniqueName.1
135         else
136             return -1                   /* SQLCommand returned empty list => userID not found! */
137
138     ::METHOD getUserPwd
139         ---------------------------
140         /*  Returns the password
141             Arguments:  uID         = UserID
142             Returns:    -1          on DB-Access Error
143                         -2          when userID not found
144                         password    when there is one       */
145         ---------------------------
146         parse arg uID
147         cmd = "select password from user where uID="||d2c(39)||uID||d2c(39)||";"
148         res = SQLCommand(SQLout,cmd)
149         if res \= 0 then
150             return -1
151         if SQLout.password.0 then       /* SQLout.password.0 = Resultcount of SQLCommand: 1=TRUE=found! */
152             return SQLout.password.1
```

```
153         else
154             return -2                   /* SQLCommand returned empty list => userID not found! */
155
156     ::METHOD setUserPwd
157         ----------------------------
158         /*  Sets/Changes user password
159             Arguments:  uID         = userID
160                         passwd      = users password
161             Returns:    <0          error
162                         0           password changed        */
163         ----------------------------
164         parse arg uID,passwd
165         if passwd = "-1" then
166             return -2                           /* BAD IDEA, -1 if a flag value... */
167         cmd = "update user set password="||d2c(39)||passwd||d2c(39)||" where uID="||d2c(39)||uID||d2c(39)||";"
168         res = SQLCommand(SQLout,cmd)
169         if res \= 0 then
170             return -1                           /* some DB-Access Error */
171         passwdtest = self~getUserPwd(uID)
172         if passwdtest = -1 then
173             return -3                           /* User does not exist */
174         else
175             if passwdtest \= passwd then
176                 return -4                       /* Password has not been changed for some unknown reason */
177         return 0
178
179     ::METHOD createSID
180         ----------------------------
181         /*  Creates a session ID and set the expire time n minutes into the future, where
182             n is retrieved from the local environment:
183             n = .local~sessionTimeOut       (defined at the beginning of this script)
184             Arguments:  uID         = User ID
185             Returns:    sID         = a session identifier, valid for n minutes     */
186         ----------------------------
187         parse arg uID
188         tme = time("L")
189         parse var tme "." t                 -- t = microseconds of system time (6 Digits)
190         r = random(1000,9999,t)             -- r = random number between 1000 and 9999, using t as random seed
```

```
191         sID = r||uID||t                     -- complete SID is a concatination of t,r and userID
192         cmd = "insert into session values("||d2c(39)||sID||d2c(39)||","
193         cmd = cmd||d2c(39)||uID||d2c(39)||",DATE_ADD(NOW(),INTERVAL "||.sessionTimeOut||" MINUTE));"
194         res = SQLCommand(SQLout,cmd)
195         if res = 0 then
196             return sID
197         return -1                           -- some error, don't know what, this is not supposed to happen
198
199     ::METHOD findUID PRIVATE
200         ---------------------------
201         /*  Finds out who owns a specific SID but does not change the session expire (like getUID4SID)
202             Arguments:  checkSID    = session ID
203             Returns:    uID         = userID for user who owns the session (if still valid)
204                         -1          on error
205                         -2          session not found, probably invalid */
206         ---------------------------
207         parse arg checkSID
208         cmd = "select uID from session where (sID="
209         cmd = cmd||d2c(39)||checkSID||d2c(39)||") and (expireTime > now());"
210         res = SQLCommand(SQLout,cmd)
211         if res \= 0 then
212             return -1                           /* some error while DB-Access */
213         if SQLout.uID.0 = 0 then
214             return -2                           /* Session not valid (either no such sID or session timed out) */
215         return SQLout.uID.1                     /* return userID if session still valid */
216
217     ::METHOD getUID4SID
218         ---------------------------
219         /*  Finds out who owns a specific SID and prolongs the session expire
220             Arguments:  checkSID    = session ID
221             Returns:    uID         = userID for user who owns the session (if still valid)   */
222         ---------------------------
223         parse arg checkSID
224         uid = self~findUID(checkSID)
225         if uid < 0 then
226             return uid                                  /* error, see findUID for return value details*/
227         res = self~updateSessionExpire(checkSID)        /* increase session expire */
228         return uid
```

```
229
230     ::METHOD closeSession
231         ----------------------------
232         /*  Deletes Session from database after logout (or timeout)
233             Arguments:  sessionID   = sessionID of session to delete
234             Returns:    -1          on error
235                         0           on success          */
236         ----------------------------
237         parse arg sessionID
238         cmd = "delete from session where sID="||d2c(39)||sessionID||d2c(39)||";"
239         res = SQLCommand(SQLout,cmd)
240         if res \=0 then
241             return -1
242         return 0
243
244     ::METHOD updateSessionExpire
245         ----------------------------
246         /*  When this method is invoked, the timeout for the specified session is
247             postponed (.local~sessionTimeOut) minutes into the future
248             Arguments:  sessionID   = session identifier of session to be prolonged
249             Returns:    -1          on error
250                         0           on success              */
251         ----------------------------
252         parse arg sessionID
253         checkSession = self~findUID(sessionID)
254         if checkSession < 0 then                /* Session does not exist or already expired */
255             return checkSession                 /* or unable to query DB */
256         cmd = "update session set expireTime=DATE_ADD(NOW(),INTERVAL "||.sessionTimeOut
257         cmd = cmd||" MINUTE) where sID="||d2c(39)||sessionID||d2c(39)||";"
258         res = SQLCommand(SQLout,cmd)
259         if res \= 0 then
260             return -1                           /* some error while DB-Access */
261         return 0
262
263     ::METHOD deleteAllExpiredSID
264         ----------------------------
265         /*  Deletes all sessions from the database that are expired by now.
266             Sessions may remain in the database, when user does not log himself out properly,
```

```
267             so this should be done from time to time.
268             Arguments:  none at all
269             Returns:    0           on success
270                         else        on error                        */
271         ----------------------------
272         cmd = "delete from session where expireTime<now();"
273         res = SQLCommand(SQLout,cmd)
274         return
275
276     ::METHOD createEvent
277         ----------------------------
278         /*  Insert an event (incl. optional properties) for a specific user into the database
279             Arguments:  userID
280                         categoryID              number according to the category definition table
281                         eventDescription        not more than 100 characters
282                         eventDate               in MySQL-Date-Format: 'yyyy-mm-dd'
283                         eventTime               in MySQL-Time-Format: 'hh:mm:ss' (seconds must not be omitted!)
284                         eventDuration           same Time-Format as above
285             Returns:    0                       success
286                         else                    error               */
287         ----------------------------
288         parse arg userID,categoryID,eventDescription,eventDate,eventTime,eventDuration
289         cmd = "insert into event values("||copies(d2c(39),2)||","||d2c(39)||userID||d2c(39)||","
290         cmd = cmd||categoryID||","||d2c(39)||eventDescription||d2c(39)||","
291         cmd = cmd||d2c(39)||eventDate||d2c(39)||","||d2c(39)||eventTime||d2c(39)||","
292         cmd = cmd||d2c(39)||eventDuration||d2c(39)||");"
293         res = SQLCommand(SQLout,cmd)
294         return res                              /* res=0 => success, else failure */
295
296     ::METHOD deleteEvent
297         ----------------------------
298         /*  Delete specified event
299             Arguments:  eventID         event to be deleted
300             Return:     0               allright
301                         other           some error          */
302         ----------------------------
303         parse arg eventID
304         cmd = "delete from event where eventID="||d2c(39)||eventID||d2c(39)||";"
```

```
305         res = SQLCommand(SQLout,cmd)
306         return res                  /* res=0 => success, else failure */
307
308     ::METHOD updateEvent
309         ------------------------------------------------------------------------
310         /*  Change event properties. No argument may be omitted.
311             Arguments:  eventID
312                         categoryID              number according to the category definition table
313                         eventDescription        not more than 100 characters
314                         eventDate               in MySQL-Date-Format: 'yyyy-mm-dd'
315                         eventTime               in MySQL-Time-Format: 'hh:mm:ss' (seconds may not be omitted!)
316                         eventDuration           same Time-Format as above
317             Returns:    0                       OK
318                         else                    Failure             */
319         ------------------------------------------------------------------------
320         parse arg eventID,categoryID,eventDescription,eventDate,eventTime,eventDuration
321         cmd = "update event set categoryID="||d2c(39)||categoryID||d2c(39)||", "
322         cmd = cmd||"description="||d2c(39)||eventDescription||d2c(39)||", "
323         cmd = cmd||"eventDate="||d2c(39)||eventDate||d2c(39)||", "
324         cmd = cmd||"eventTime="||d2c(39)||eventTime||d2c(39)||", "
325         cmd = cmd||"duration="||d2c(39)||eventDuration||d2c(39)
326         cmd = cmd||" where eventID="||d2c(39)||eventID||d2c(39)||";"
327         res = SQLCommand(SQLout,cmd)
328         return res                   /* res=0 => success, else failure */
329
330     ::METHOD retrieveEvent
331         ---------------------------
332         /*  Returns all info about a specific event in a Rexx Stem Variable (Array)
333             Arguments:  eventID         Unique identifier of event
334             Returns:    SQLout.         Array with all columns of the event the database supplies */
335         ---------------------------
336         parse arg eventID
337         cmd = "select * from event where eventID="||d2c(39)||eventID||d2c(39)||";"
338         res = SQLCommand(SQLout,cmd)
339         return SQLout.
340
341     ::METHOD findEvent
342         ---------------------------
```

```
343         /*  Find all events for a user that matches a specific category, could be useful someday...
344             Arguments:  userID
345                         categoryID      identifier of the category
346
347             Returns:    SQLout.         Array with all events of a category  */
348         ---------------------------
349         parse arg userID,categoryID
350         cmd = "select eventID from event where categoryID="||d2c(39)||categoryID||d2c(39)
351         cmd = cmd||" and uID="||d2c(39)||userID||d2c(39)||" order by eventID;"
352         res = SQLCommand(SQLout,cmd)
353         return SQLout.
354
355     ::METHOD getEventsOfDay
356         ---------------------------
357         /*  Returns all eventIDs of a specific date and user
358             Arguments:  userID
359                         eventDt                     date of event
360
361             Returns:    SQLout.                     Array with all events of one day
362                         SQLout.eventID.0=0          when no event found                 */
363         ---------------------------
364         parse arg userID,eventDt
365         cmd = "select eventID,eventTime,description from event where eventDate="||d2c(39)||eventDt||d2c(39)
366         cmd = cmd||" and uID="||d2c(39)||userID||d2c(39)||" order by eventTime;"
367         res = SQLCommand(SQLout,cmd)
368         if SQLout.eventID.0 = 'SQLOUT.EVENTID.0' then SQLout.eventID.0 = 0
369         return SQLout.
370
371     ::METHOD getCategory
372         ---------------------------
373         /*  Returns the name of a given categoryID
374             Arguments:  categoryID
375
376             Returns:    SQLout.category.1           Name of category
377                         -1                          DB-Error
378                         -2                          no such category          */
379         ---------------------------
380         parse arg catID
```

```
381         cmd = "select category from category where categoryID="||catID||";"
382         res = SQLCommand(SQLout,cmd)
383         if res \=0 then return res
384         if SQLout.category.0 = 0 then return -2
385         return SQLout.category.1
386
387     ::METHOD getMaxCategory
388         ---------------------------
389         /*  Returns the name of a given categoryID
390             Arguments:  none
391
392             Returns:    SQLout.category.1           max. Indexnumber of category
393                         -1                          DB-Error                    */
394         ---------------------------
395         cmd = "select max(categoryID) as mxid from category;"
396         res = SQLCommand(SQLout,cmd)
397         if res \=0 then return res
398         return SQLout.mxid.1
399
```