```
/* m3sql.cls -- classes structured after EIA/CDIF-M3, CTI --> SQL */

:: REQUIRES rxSQLutil.cmd       /* Mark Hessling's RexxSQL routines     */
:: REQUIRES sort_Util.cmd       /* get access to sort-routines          */

/* ---------------------------------------------------------------------------- */
:: CLASS MO     PUBLIC

:: METHOD INIT CLASS

   CALL m3sql.debug .false    /* set debug-variable   */

        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR", "MO_TYPE", "ALIASES", ,
                        "CDIFMetaIdentifier", "Constraints", "Description",
                        "Name", "LongName", "Usage"   )

   self ~ table_name = "MO_"    /* Base table name      */
   CALL make_sql_stmts self     /* create SQL-statements*/


:: METHOD attrib_list CLASS ATTRIBUTE   /* list of MMAs                                */
:: METHOD dt_stem     CLASS ATTRIBUTE   /* stem-array of datatypes                     */
:: METHOD sql_insert  CLASS ATTRIBUTE   /* insert statement                            */
:: METHOD sql_select  CLASS ATTRIBUTE   /* select statement (for retrieving data types
                                           dynamically */
:: METHOD stmt_insert CLASS ATTRIBUTE   /* name of prepared insert-statement           */
:: METHOD stmt_select CLASS ATTRIBUTE   /* name of prepared insert-statement           */
:: METHOD table_name  CLASS ATTRIBUTE   /* table name, also used as the cursor name    */

:: METHOD prep_insert CLASS             /* connect and prepare insert statement */
   EXPOSE sql_insert sql_select table_name stmt_select stmt_insert

   stmt_insert = table_name || "I"      /* create unique statement name for insertions  */
   IF sqlprepare( stmt_insert , sql_insert ) < 0 THEN
               rxSQL.Abort('prepare:' stmt_insert , , sqlca.)

        /* create datatype definition stem dynamically, using the SELECT-statement with
           the same order of columns as the INSERT-statement */
   stmt_select = table_name || "S"      /* create unique statement name for selects     */
   IF sqlprepare( stmt_select , sql_select  ) < 0 THEN
               rxSQL.Abort('prepare:' pp( stmt_select ), , sqlca.)

   If sqldescribe( stmt_select , "AHA"  ) < 0 Then
               rxSQL.Abort('describe:' pp( stmt_select ), , sqlca.)

   new_dt. = ""
   DO i = 1 TO AHA.COLUMN.TYPE.0
      new_dt.i = aha.COLUMN.TYPE.i
   END

   new_dt.0 = ( i - 1 )                 /* # of elements        */
   self ~ dt_stem = new_dt.             /* assign data-type stem-array to class variable */


:: METHOD dispose     CLASS             /* dispose insert- and select-statement */
   EXPOSE stmt_insert stmt_select

   IF sqlDispose( stmt_insert ) < 0 THEN rxSQL.Abort('dispose:' stmt_insert , , sqlca.)

        /* create datatype definition stem dynamically, using the SELECT-statement with
           the same order of columns as the INSERT-statement */
   IF sqlDispose( stmt_select ) < 0 THEN rxSQL.Abort('dispose:' pp( stmt_select ), , sqlca.)


:: METHOD insert   CLASS                /* insert a record into RDBMS   */
   USE ARG valueDir
    /* make sure that top-down inserts take place, otherwise FK-constraints are violated !*/
   RETURN insertRow( valueDir, .mo )
```

```
/* ... continued, belongs to class MO ... */
/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                          /* Directory to contain */
                                           /* set the object variables from directory      */
   DO item OVER .mo ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value*/
   END
                                           /* Define Attribute methods     */
:: METHOD ALIASES                ATTRIBUTE
:: METHOD CDIFMetaIdentifier     ATTRIBUTE
:: METHOD Constraints            ATTRIBUTE
:: METHOD Description            ATTRIBUTE
:: METHOD LongName               ATTRIBUTE
:: METHOD MO_TYPE                ATTRIBUTE
:: METHOD Name                   ATTRIBUTE
:: METHOD SURR                   ATTRIBUTE
:: METHOD Usage                  ATTRIBUTE

:: METHOD SortValue                        /* Value to sort after          */
   RETURN self ~ Name


/* ---------------------------------------------------------------------------- */
:: CLASS SA                SUBCLASS MO     PUBLIC

:: METHOD INIT CLASS
        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR", "ShortHand" , "CDIFNumber", "VersionNumber" )
   self ~ table_name = "SA_"    /* Base table name      */
   CALL make_sql_stmts self


:: METHOD insert   CLASS                   /* insert a record into RDBMS   */
   EXPOSE top
   USE ARG valueDir

   valueDir ~ MO_TYPE = "SA"               /* set type information         */
     /* make sure that top-down inserts take place, otherwise FK-constraints are violated!*/
   FORWARD CLASS( super ) CONTINUE

   IF RESULT THEN                          /* "RESULT" contains result of last return value  */
      RETURN insertRow( valueDir, .sa )
   ELSE RETURN .false

/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                          /* Directory to contain */
                                           /* set the object variables from directory      */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .sa ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value*/
   END
                                           /* make sure that an SA has a long name */
   IF self ~ LongName = .nil THEN self ~ LongName = self ~ Name

                                           /* Define Attribute methods     */
:: METHOD CDIFNumber             ATTRIBUTE      /* official EIA/CDIF-number of standard  */
:: METHOD SURR                   ATTRIBUTE
:: METHOD ShortHand              ATTRIBUTE      /* abbreviation used while developing SA */
:: METHOD VersionNumber          ATTRIBUTE
```

```
/* ------------------------------------------------------------------------------ */
:: CLASS CMO              SUBCLASS MO     PUBLIC

:: METHOD INIT CLASS
        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR" )
   self ~ table_name = "CMO_"    /* Base table name      */
   CALL make_sql_stmts self

:: METHOD insert   CLASS                /* insert a record into RDBMS   */
   USE ARG valueDir
     /* make sure that top-down inserts take place, otherwise FK-constraints are violated! */
   FORWARD CLASS( super ) CONTINUE
   IF RESULT THEN                       /* "RESULT" contains result of last return value   */
      RETURN insertRow( valueDir, .cmo )
   ELSE RETURN .false


/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                       /* Directory to contain */
                                        /* set the object variables from directory      */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .cmo ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value */
   END
   self ~ bDefinedInSA = ( tmpDir ~ bDefinedInSA )      /* explicitly used in SA        */

        /* Define Attribute methods     */
:: METHOD SURR                  ATTRIBUTE
:: METHOD bDefinedInSA          ATTRIBUTE


/* ------------------------------------------------------------------------------ */
:: CLASS MA               SUBCLASS CMO    PUBLIC

:: METHOD INIT CLASS
   EXPOSE attrib_list

        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR", "DataType", "Domain", "IsOptional", "Length" )
   self ~ table_name = "MA_"    /* Base table name      */
   CALL make_sql_stmts self

:: METHOD insert   CLASS                /* insert a record into RDBMS   */
   USE ARG valueDir

   valueDir ~ MO_TYPE = "MA"            /* set type information         */
     /* make sure that top-down inserts take place, otherwise FK-constraints are violated!*/
   FORWARD CLASS( super ) CONTINUE
   IF RESULT THEN                       /* "RESULT" contains result of last return value  */
      RETURN insertRow( valueDir, .ma )
   ELSE RETURN .false


/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                       /* Directory to contain */
                                        /* set the object variables from directory      */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .ma ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value */
   END

   self ~ BelongsTo = .nil              /* initialise to .nil   */
                                        /* Define Attribute methods     */
:: METHOD BelongsTo             ATTRIBUTE       /* points to AMO, MA belongs to */
:: METHOD DataType              ATTRIBUTE
:: METHOD Domain                ATTRIBUTE
:: METHOD IsOptional            ATTRIBUTE
:: METHOD Length                ATTRIBUTE
:: METHOD SURR                  ATTRIBUTE

:: METHOD SortValue                     /* Value to sort after          */
   RETURN self ~ BelongsTo ~ LongName || ":" self ~ name
```

```
/* ------------------------------------------------------------------------------ */
:: CLASS AMO              SUBCLASS CMO    PUBLIC

:: METHOD INIT CLASS
        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR" )
   self ~ table_name = "AMO_"   /* Base table name      */
   CALL make_sql_stmts self

:: METHOD insert   CLASS                /* insert a record into RDBMS   */
   USE ARG valueDir

   IF valueDir ~ MO_TYPE = .nil THEN    /* RootObject in hand ?         */
      valueDir ~ MO_TYPE = "AMO"        /* set type information         */

     /* make sure that top-down inserts take place, otherwise FK-constraints are violated! */
   FORWARD CLASS( super ) CONTINUE
   IF RESULT THEN                       /* "RESULT" contains result of last return value   */
      RETURN insertRow( valueDir, .amo )
   ELSE RETURN .false

/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                       /* Directory to contain */
                                        /* set the object variables from directory      */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .amo ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value */
   END
   self ~ SubtypeOf = tmpDir ~ SubtypeOf
                                        /* make sure that all AMOs have a long name     */
   IF self ~ LongName = .nil THEN self ~ LongName = self ~ Name
   self ~ localMA = .relation ~ new     /* create an empty relation for local MAs       */

                                        /* Define Attribute methods     */
:: METHOD LocalMA               ATTRIBUTE       /* LocalMA[ name ] = MA-object  */
:: METHOD SURR                  ATTRIBUTE
:: METHOD SubtypeOf             ATTRIBUTE       /* string of Supertypes   */

:: METHOD SortValue                     /* Value to sort after          */
   RETURN self ~ LongName
/* ------------------------------------------------------------------------------ */
:: CLASS ME               SUBCLASS AMO    PUBLIC

:: METHOD INIT CLASS
        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR", "Type" )
   self ~ table_name = "ME_"   /* Base table name      */
   CALL make_sql_stmts self

:: METHOD insert   CLASS                /* insert a record into RDBMS   */
   USE ARG valueDir

   valueDir ~ MO_TYPE = "ME"            /* set type information         */
     /* make sure that top-down inserts take place, otherwise FK-constraints are violated !*/
   FORWARD CLASS( super ) CONTINUE
   IF RESULT THEN                       /* "RESULT" contains result of last return value   */
      RETURN insertRow( valueDir, .me )
   ELSE RETURN .false

/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                       /* Directory to contain */
                                        /* set the object variables from directory      */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .me ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value */
   END
                                        /* Define Attribute methods     */
:: METHOD SURR                  ATTRIBUTE
:: METHOD Type                  ATTRIBUTE
```

```
/* ------------------------------------------------------------------------------ */
:: CLASS MR              SUBCLASS AMO    PUBLIC

:: METHOD INIT CLASS
        /* define list of attributes */
   self ~ attrib_list = .list ~ of( "SURR", "MinSourceCard", "MaxSourceCard", ,
                                    "MinDestCard"  , "MaxDestCard" )
   self ~ table_name = "MR_"   /* Base table name      */
   CALL make_sql_stmts self

:: METHOD insert   CLASS               /* insert a record into RDBMS   */
   USE ARG valueDir

   valueDir ~ MO_TYPE = "MR"           /* set type information         */
     /* make sure that top-down inserts take place, otherwise FK-constraints are violated!*/
   FORWARD CLASS( super ) CONTINUE
   IF RESULT THEN                      /* "RESULT" contains result of last return value  */
      RETURN insertRow( valueDir, .mr )
   ELSE RETURN .false

/* ------- -------- INSTANCE methods -------- -------- */
:: METHOD INIT
   USE ARG tmpDir                      /* Directory to contain */
                                       /* set the object variables from directory       */
   FORWARD CLASS ( super ) CONTINUE
   DO item OVER .mr ~ attrib_list
      .message ~ new( self, item || "=", "I", tmpDir ~ entry( item ) ) ~ send /* set value*/
   END

                                       /* Define Attribute methods     */
:: METHOD MinSourceCard         ATTRIBUTE
:: METHOD MaxSourceCard         ATTRIBUTE
:: METHOD MinDestCard           ATTRIBUTE
:: METHOD MaxDestCard           ATTRIBUTE
:: METHOD SURR                  ATTRIBUTE
```

```
/* ------------------------------------------------------------------------------ */
/* inserts the appropriate (determined by clsObject) fragment into the RDBMS;
   returns .false, if MO exists already, .true if insertions are o.k.   */
/* insert row into database     */
:: ROUTINE insertRow
   USE ARG tmpObject, clsObject

continue:                       /* address, if CDIFMetaIdentifier is created    */
   IF clsObject = .mo THEN      /* check on first invocation per tmpObj only !  */
   DO
      tmpCID = ( tmpObject ~ CDIFMetaIdentifier )
      /* checking for duplicate CDIFMetaIdentifier */
      sql_check  = "SELECT SURR, CDIFMETAIDENTIFIER, NAME, MO_TYPE, LONGNAME FROM MO_",
                   "WHERE CDIFMETAIDENTIFIER = '" || tmpCID || "'"
      IF sqlcommand( "query", sql_check ) < 0 THEN rxSQL.Abort(sql_check, , sqlca.)
      rowNum = query.SURR.0     /* get # of retrieved records           */

      IF rowNum > 0 THEN        /* MO exists already            */
      DO
         tmpMO_Type = tmpObject ~ mo_type
         tmpName    = tmpObject ~ name

         DO i = 1 TO query.SURR.0     /* loop, see whether duplicate CDIFMetaIdentifier */
            IF query.MO_TYPE.i = tmpMO_Type THEN        /* O.K., of same type   */
            DO
               IF ( POS( tmpMO_Type, "ME MR AMO") > 0 ) THEN    /* really same MO ?     */
                  bReuse = ( query.LONGNAME.i = tmpObject ~ Longname )
               ELSE IF tmpMO_Type = "MA" THEN           /* oh, an MA-duplicate          */
               DO  /* get AMO this MA belongs to */
                   tmpAMO = .xfer_cdif ~ IsLocalMetaAttributeOf[ tmpObject ]
                   /* does MA exist and is it assigned to the same AMO ?               */
                    sql_check_amo = ,   /* SQL-query    */
"SELECT SURR, CDIFMETAIDENTIFIER, NAME, MO_TYPE, LONGNAME "                 ,
"FROM   AMO "                                                               ,
"WHERE      "                                                               ,
"       CDIFMETAIdentifier = '" || tmpAmo ~ CDIFMetaIdentifier || "'" ,
"       AND"                                                                ,
"       LONGNAME           = '" || tmpAmo ~ LongName           || "'" ,
"       AND"                                                                ,
"       SURR IN ( SELECT DESTINATION FROM   IsLocalMetaAttributeOf_ " ,
"                 WHERE  SOURCE IN ( SELECT SURR FROM MA"                   ,
"                                    WHERE  CDIFMETAIdentifier = '" || tmpCID || "' AND" ,
"                                           NAME = '" || tmpName || "' ) ) "

                   IF sqlcommand( "queryAMO", sql_check_amo ) < 0 THEN
                      rxSQL.Abort(sql_check_amo, , sqlca.)
                   bReuse = ( queryAMO.SURR.0 > 0 )     /* if AMO exists, then reuse MA */
               END
               ELSE     /* an SA in hand        */
                  bReuse = ( query.NAME.i = tmpName )

               IF bReuse THEN           /* o.k., inferring: same MO, hence reuse it     */
               DO
                  tmpObject ~ surr = query.SURR.1 /* use SURR of MO as stored in RDBMS  */
                  SAY "              (MO exists already, gets reused)"
                  RETURN .false       /* no insertion needed, object exists already    */
               END
            END
         END
         tmpString = ""
         IF ( POS( tmpMO_Type, "ME MR AMO") > 0 ) THEN
            tmpString = "LongName" pp( tmpObject ~ longname )
         .error ~ SAY                   /* empty line to STDERR */
         .error ~ SAY( "..." "Duplicate CDIFMetaIdentifier:" pp( tmpCID ) )
      END
   END
   stmt = clsObject ~ stmt_insert       /* get appropriate prepared insert-statement    */
   DV.  = ""                            /* stem to contain values                       */
   i = 0                                /* fill value-stem                              */
   DO item OVER clsObject ~ attrib_list /* iterate in the attrib_list order             */
      i = i + 1                         /* assign values for bind variables             */
      dv.i = check4null( .message ~ new( tmpObject, item ) ~ send ) /* get & set value  */
   END
   dv.0 = i           /* Insert row into table: */
   IF sqlexecute( stmt , ".", "DV." ) < 0 THEN rxSQL.Abort(stmt, , sqlca.)
   RETURN .true       /* indicate that insertions along the inheritance tree is o.k.    */
```

```
/* ---------------------------------------------------------------------------- */
   /* routine to create attribute methods and the appropriate sql-insert statement */
:: ROUTINE make_sql_stmts
   USE ARG object

   column = ""                  /* columns of SQL-insert statement      */
   values = ""                  /* values of SQL-insert statement       */

   i = 0        /* create and set attribute methods for passed in attributes */
   DO item OVER object ~ attrib_list
      i = i + 1
      IF column = "" THEN column = item
                     ELSE column = column "," item

      IF values = "" THEN values = ":" || i             /* build bind variables */
                     ELSE values = values ", :" || i
   END
               /* create and save insert-SQL-statement with object     */
   object ~ sql_insert = ,
            "INSERT INTO" object ~ table_name "(" column ") VALUES (" values ") "

               /* create and save select-SQL-statement with object     */
   object ~ sql_select = "SELECT" column "FROM" object ~ table_name "WHERE SURR = :1"
   RETURN

/* ---------------------------------------------------------------------------- */
:: ROUTINE pp; RETURN "[" || ARG( 1 ) || "]"

/* ---------------------------------------------------------------------------- */
/* return the next unique surrogate value       */
:: ROUTINE NEXT_SURR    PUBLIC

   IF .m3sql.surr = ".M3SQL.SURR" THEN      /* not set in .local as of yet  */
   DO
      /* get highest value of SURR out of table MO and all tables representing MMR */
      query = 'SELECT MAX( SURR ) "MAX" FROM mo                      UNION'  ,
              'SELECT MAX( SURR ) "MAX" FROM IsUsedIn_               UNION'  ,
              'SELECT MAX( SURR ) "MAX" FROM IsLocalMetaAttributeOf_ UNION'  ,
              'SELECT MAX( SURR ) "MAX" FROM HasSubtype_             UNION'  ,
              'SELECT MAX( SURR ) "MAX" FROM HasSource_              UNION'  ,
              'SELECT MAX( SURR ) "MAX" FROM HasDestination_              '  ,
              'ORDER BY 1 DESC                                            '

      IF sqlcommand( "tmp", query ) < 0 THEN rxSQL.Abort('command: executing', , sqlca.)

      IF DATATYPE( tmp.max.1, "W" ) THEN
         .local ~ m3sql.surr = tmp.max.1 + 1  /* save to .local       */
      ELSE      /* no values so far, start out with "1"         */
         .local ~ m3sql.surr = 1              /* save to .local       */
   END
   ELSE
      .local ~ m3sql.surr = .m3sql.surr + 1   /* save to .local       */

   RETURN .m3sql.surr                         /* return new highest value     */
```