```
/************************************************************

docbegin:

*  Name of source file:         j_src_lc_work.c

*  Version of source file:      0.01

*  Parent component:            None

*  Programmer:                  Stefan Larsson

*  Affiliation:                 Stockholm Observatory

*  Contact address:             Stockholm Observatory
*                               SE-133 36 Saltsjobaden, Sweden
*                               Phone: +46 8-164464  FAX: +46 8-7174719
*                               stefan@astro.su.se

*  Purpose:                     Extract spectra and light curves for individual
*                               sources in one science window of imaging data.

*  Origin date:                 000511

*  Update history:              0.01  000511 Adaption of test code jextr5.c

*  Type of element:             Executable

*  Part of package:             j_src_lc

*  Name:                        j_src_lc_work

*  Calls:

*  Returns:

*  Required components:         DAL, PIL, RIL, DAL3GEN, DAL3JEMX, DAL3HK


docend:

Various remarks:

This version disregards the OBTime and makes no selection based on
time intervals and not on GTIs.


Flow of program:

Error codes:

Basic offset:
J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START + J_SLC_ERROR_START


J_SLC_WRONG_JEMX_NUM (-1):  Called with illegal value of jemxNum
J_SLC_NOT_IN_SMODE   (-2):  Started in other than ISDC_SINGLE_MODE
J_SLC_WRONG_JEMX_NUM (-3):  inconsistency between numbers of members
J_SLC_ROWID_ERROR    (-4):  Problem with rowidentifications
J_SLC_WRONG_TYPE     (-6):  Wrong event type for srce extraction
J_SLC_NO_EVTS_SELECTED (-7): No events matched selection


*****************************************************************/

#include <j_src_lc.h>
```

```
int j_src_lc_work(
                    dal_element *ptr_inSWG,         /*  Pointer to input SWG  */
                    JEMX_num jemxNum,               /*  Number of JEMX unit   */
                    JEMX_type jemxType,             /*  Type of events        */
                    int nChanBins,         /*  Number of energy channels     */
                    int *chanLow,           /*  Lower energy channel boundary */
                    int *chanHigh,          /*  Upper energy channel boundary */
                    int chatty,                     /*  Level of debugging output  */
                    int status)                     /*  Status flag           */
{

  /*  Ordinary variables  */

  void *arrayPointer = NULL;

  void *buff_mode    = NULL;
  void *buff_greyFilter = NULL;
  void *buff_greyOBTime = NULL;
  void *buff_deadTime = NULL;
  void *buff_deadOBTime = NULL;
  void *buff_detx    = NULL;
  void *buff_dety    = NULL;
  void *buff_pi      = NULL;
  void *buff_evteval = NULL;
  void *buff_timeOBT = NULL;

  unsigned char *pi;
  unsigned char *evteval;
  char gtiNames[14];


  char nameStatus[14];   /* Name of instrument status JMXi-INST-STA */
  char nameDeadTime[14];  /* Name of dead time history JMXi-DEAD-STA */
  char evtsshdName[14];
  char shadidxName[14];
  char typecorName[14];
  char typeprpName[14];
  char tableName[14];
  char strLong[20];
  char unit[30];
  char comment[80];
  char RowFilter[100];
  char strjemxType[5];

  int i, j;
  int numMembers;
  int NumFound;
  int maxShadowgram;
  int minShadowgram;
  int NumIntervals;
  int iCell;           /*  Index value of cell where the event belongs  */
  int jCell;           /*  Index value of cell where the event belongs  */
  int numbGTIs;

  long loop;
  long NumValues;
  long numberOfCounts;
  long axes[DAL_MAX_ARRAY_DIMENSION];
  long NumEvents;          /*  Number of selected events  */
  long NumEventsD3J;       /*  Number of selected events from DAL3GJEMX */
  long integrationTime = 1200L;
  long iTimeBin;
  long NumGreyChanges = 0;  /* Number of grey filter changes */
  long NumGreyValues = 0;
  long NumDeadTimes = 0; /* Number of dead time */
```

```
  long NumDeadTimeValues = 0;

  /*
   *   NB!  This is a temporary solution that must be changed
   */
  /* for test purpose: */
  long jm1[4], jm2[4];


  float *detx;       /*  Pointer to array with DETX values of events */
  float *dety;       /*  Pointer to array with DETY values of events */
  float xCellCenter; /*  x coord. of center of cell where event belongs */
  float yCellCenter; /*  y coord. of center of cell where event belongs */

  short *mask;       /*  Pointer to mask array. */

  double sourcePosX[J_SRC_MAXSRC]; /* Source X-coord. src# 0,1,2... */
  double sourcePosY[J_SRC_MAXSRC]; /* Source Y-coord. src# 0,1,2... */
  double effArea;
  double resSize = 0.2e0L;

  /*  DAL, RIL, PIL, COMMON declarations  */

  dal_element  *ptrDS        = NULL;  /*  Pointer to chosen DS            */
  dal_element  *ptrGrey      = NULL;
  dal_element  *ptrDTT       = NULL;
  dal_element  *ptrTimeDS    = NULL;


  dal_dataType DALtype;

  dal_accessType   accessType;

  dal_baseType     baseType;

  OBTime          *timeOBT = NULL;
  OBTime          *gtiStart = NULL;
  OBTime          *gtiEnd = NULL;
  OBTime           OBTstart[1];
  OBTime           OBTend[1];
  OBTime           startTime = 0;
  OBTime           endTime = 0;

  Instrument instrument = JEMX1;

  /*  End of DAL etc. declarations  */



  /*  Immediate return if called while executable is in error condition  */

  if( status != ISDC_OK ) return( status );


  /*
   *
   *   Set up data stucture names to match jemx type
   *
   */

  if( jemxNum == 1 ){
    strcpy( evtsshdName,"JMX1-EVTS-SHD" );
    strcpy( shadidxName,"JMX1-SHAD-IDX" );
    strcpy( nameStatus,"JMX1-INST-STA" );
    strcpy( nameDeadTime,"JMX1-DEAD-STA" );
    if( jemxType == FULL ){
      strcpy( typecorName,"JMX1-FULL-COR" );
      /*   strcpy( typeprpName,"JMX1-FULL-PRP" ); */
    } else if( jemxType ==REST ){
      strcpy( typecorName,"JMX1-REST-COR" );
    }
    else{
      status = RILlogMessage( NULL,Error_1,
                              "Can not handle jemx type %i",jemxType,status );
    }
  } else if( jemxNum == 2 ){
    strcpy( evtsshdName,"JMX2-EVTS-SHD" );
    strcpy( shadidxName,"JMX2-SHAD-IDX" );
    strcpy( nameStatus,"JMX2-INST-STA" );
    strcpy( nameDeadTime,"JMX2-DEAD-STA" );
    if( jemxType == FULL ){
      strcpy( typecorName,"JMX2-FULL-COR" );
      /*  strcpy( typeprpName,"JMX2-FULL-PRP" );     */
    } else if( jemxType == REST ){
      strcpy( typecorName,"JMX2-REST-COR" );
    }
    else{
      status = RILlogMessage( NULL,Error_1,
                              "Can not handle jemx type %i",jemxType,status );
    }
  }
  else{
    status = RILlogMessage( NULL,Error_1,
                            "JEM-X instrument number %i impossible!",jemxNum,status );
  }

  if( jemxType == FULL ) {
    strcpy( strjemxType,"FULL" );
  } else if( jemxType == REST ) {
    strcpy( strjemxType,"REST" );
  } else {
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_WRONG_TYPE;
    return( status );
  }

  /*
   *
   *   Find appropriate DS with corrected events
   *
   */

  status = DALobjectFindElement( ptr_inSWG,typecorName,
                                 &ptrDS,status );
  if( status == DAL_ELEMENT_NOT_FOUND ){
    status = RILlogMessage( NULL,Log_0,
                            "Cannot find requested data structure: %s in SWG",typecorName );
    NumEvents = 0L;
    return( status );
  }
  else if( status == ISDC_OK ){
    status = RILlogMessage( NULL,Log_0,
                            "%s is available",typecorName,status );
  }
  else return( status );


  /*
   *
   * See if output data is empty.
   *
   */
```

```
  status = DALtableGetNumRows( ptrDS,&NumEvents,status );
  if( status != ISDC_OK ) return( status );
  if( NumEvents ==  0L ){
    status = RILlogMessage(NULL,Log_0,
                           "No shadowgram is produced",status );
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_NO_EVTS_SELECTED;
    return( status );
  }
  if( chatty >= 100 ){
    status = RILlogMessage(NULL,Log_0,
                           "Number events: %d",&NumEvents );
  }

  /*  status = DALobjectFindElement( ptr_inSWG,typeprpName,&ptrTimeDS,status );
  if( status == DAL_ELEMENT_NOT_FOUND ) {

  status = RILlogMessage( NULL,Log_0,"work ##0## Cannot find requested DS: %s",typeprpName )
;

    NumEvents = 0L;
    return( status );

  } else if( status == ISDC_OK ) {
    status = RILlogMessage( NULL,Log_0,"work ##1## JMX1-FULL-PRP is avaiable");
    } else return( status ); */




  /*  For testing DAL3JEMX library; show number of events in group  */

  status = DAL3JEMXshowAllEvents( ptr_inSWG,jm1,jm2,status);
  if( status != ISDC_OK ) {
    status = RILlogMessage( NULL,Log_0,
                            "work ##2## Malfunction of DAL3JEMXshowAllEvents,status = %d",st
atus );
  }
  status = RILlogMessage( NULL,Log_0,
                          "work ##3## Evts JMX1: FULL: %ld, REST: %ld",jm1[FULL],jm1[REST]);
  status = RILlogMessage( NULL,Log_0,
                          "work ##4## Evts JMX2: FULL: %ld, REST: %ld",jm2[FULL],jm2[REST]);

  /*
   *
   * Obtain Good-Time Interval (GTI) information.
   *
   */


  strcpy( gtiNames,"MERG" );
  if( jemxNum == 1 ){
    instrument = JEMX1;
    status = DAL3HKgetNumGTI( ptr_inSWG,instrument,&gtiNames,1,&numbGTIs,status );
    if( status != ISDC_OK ){
      status = RILlogMessage(NULL,Log_0,
                             "DAL3HKgetNumGTI failed, status = %d",status );
    }
  } else if( jemxNum == 2 ){
    instrument = JEMX2;
    status = DAL3HKgetNumGTI( ptr_inSWG,instrument,&gtiNames,1,&numbGTIs,status );
    if( status != ISDC_OK ){
      status = RILlogMessage(NULL,Log_0,
                             "DAL3HKgetNumGTI failed, status = %d ",status );
    }
  }
  else{
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_WRONG_JEMX_NUM;
    return( status );
  }


  /* Read actual good time intervals */

  startTime = DAL3_NO_OBTIME;
  endTime = DAL3_NO_OBTIME;
  strcpy( gtiNames,"MERG" );
  if( jemxNum == 1 ){
    instrument = JEMX1;
    status = DAL3HKgetGTI( ptr_inSWG,instrument,&gtiNames,1,startTime,endTime,
                           &numbGTIs,gtiStart,gtiEnd,status );
    if( status != ISDC_OK ) {
      status = RILlogMessage(NULL,Log_0,
                             "DAL3HKgetGTI failed, status = %d ",status);
    }
  }
  else if( jemxNum == 2 ){
    instrument = JEMX2;
    status = DAL3HKgetGTI( ptr_inSWG,instrument,&gtiNames,1,startTime,endTime,
                           &numbGTIs,gtiStart,gtiEnd,status );
    status = RILlogMessage(NULL,Log_0,
                           "DAL3HKgetGTI failed, status = %d ",status);
  }
  else{
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_WRONG_JEMX_NUM;
    return( status );
  }


  /*
   *
   * Obtain Grey Filter History
   *
   */


  /* Find status history and get number of grey filter changes */

  status = DALobjectFindElement( ptr_inSWG,nameStatus,&ptrGrey,status );
  if( status != ISDC_OK ){
    if (status == DAL_ELEMENT_NOT_FOUND ){
      if( chatty >= J_CHATTY_NORMAL ){
        status = RILlogMessage( NULL,Warning_2,"No %s %s",nameStatus,
                                "with mode and greyfilter data in current SWG" );
      }
      status = ISDC_OK;
      NumGreyChanges = 0;
    }
    else{
      status = RILlogMessage( NULL,Error_2,"Error in searching %s",nameStatus,status );
    }
  }
    else{
      status = DALtableGetNumRows( ptrGrey,&NumGreyChanges,status );
      if( status != ISDC_OK ){
        status = RILlogMessage( NULL,Error_2,
                                "Error in determing number of grey filter changes, status =
%d",
                                status );
```

```
        return( status );
      }
      if( NumGreyChanges ==  0L ){
        status = RILlogMessage(NULL,Log_0,
                               "No shadowgram is produced" );
        status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
          + J_SLC_ERROR_START + J_SLC_NO_EVTS_SELECTED;
        return( status );
      }
    }




  /*
   *  Allocate memory buffer for MODE, GREY_FILTER and OB_TIME
   */


  if( NumGreyChanges > 0 ){

    status = DALallocateDataBuffer( &buff_mode, NumGreyChanges*sizeof( short ),status );
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_3, "Problem allocating memory for format status",
                              status );
      return( status );
    }

    status = DALallocateDataBuffer( &buff_greyFilter,NumGreyChanges*sizeof( short ),status )
;
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_3, "Problem allocating memory for grey filter statu
s",
                              status );
      return( status );
    }

    status = DALallocateDataBuffer( &buff_greyOBTime,NumGreyChanges*sizeof( OBTime ),status
);
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_3,"Problem allocating memory for grey filter OBTime
",
                              status );
      return( status );
    }


    /* Read status and grey filter changes */

    DALtype = DAL_SHORT;
    status = DALtableGetCol( ptrGrey,"MODE",0,&DALtype,&NumGreyValues,
                             buff_mode, status );
    status = DALtableGetCol( ptrGrey,"GREY_FILTER",0,&DALtype,&NumGreyValues,
                             buff_greyFilter,status );
    status = DAL3GENtableGetOBT( ptrGrey,"OB_TIME",0,&NumGreyValues,
                             buff_greyOBTime,status );
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_2,"Error reading greyfilter & mode changes!",
                              status );
      return( status );
    }
    if( NumGreyValues != NumGreyChanges ){
      status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
        + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
      return( status );
    }
  }
  else{
    if( chatty > J_CHATTY_NORMAL ){
      status = RILlogMessage( NULL,Warning_2,
                              "No greyfilter values found; spectra will not be corrected.",
                              status );
      return( status );
    }
  }


  /*
   *
   *   Obtain Dead Time History
   *
   */

  /* Find dead time history table */

  status = DALobjectFindElement(ptr_inSWG,nameDeadTime,&ptrDTT,status);
   if( status != ISDC_OK ){
    if ( status == DAL_ELEMENT_NOT_FOUND ){
      if( chatty >= J_CHATTY_NORMAL ){
        status = RILlogMessage( NULL,Warning_2,"No %s %s",nameDeadTime,
                                "with deadtime history in current SWG" );
      }
      status = ISDC_OK;
      NumDeadTimes = 0;
    }
    else{
      status = RILlogMessage( NULL,Error_2,"Error in searching %d",nameDeadTime,status );
    }
  }
    else{
      status = DALtableGetNumRows( ptrDTT,&NumDeadTimes,status );
      if( status != ISDC_OK ){
        status = RILlogMessage( NULL,Error_2,
                                "Error in determing number of deadtime history entries, stat
us = %d",
                                status );
        return( status );
      }
      if( NumDeadTimes ==  0L ){
        status = RILlogMessage(NULL,Log_0,
                               "No shadowgram is produced" );
        status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
          + J_SLC_ERROR_START + J_SLC_NO_EVTS_SELECTED;
        return( status );
      }
    }


    /*
     *  Allocate memory buffer for DEADTIME, OB_TIME
     */


  if( NumDeadTimes > 0 ){
    status = DALallocateDataBuffer( &buff_deadTime,NumDeadTimes*sizeof( float ),status );
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_3,"Problem allocating memory for format status",
                              status );
      return( status );
    }

    status = DALallocateDataBuffer( &buff_deadOBTime,NumDeadTimes*sizeof( OBTime ),status );
```

```
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_3,"Problem allocating memory for grey filter status
",
                              status );
      return( status );
    }



    /* Read dead time history */

    DALtype = DAL_FLOAT;
    status = DALtableGetCol( ptrDTT,"DEADTIME",0,&DALtype,&NumDeadTimeValues,
                             buff_deadTime,status );
    status = DAL3GENtableGetOBT( ptrDTT,"OB_TIME",0,&NumDeadTimeValues,
                             buff_deadOBTime,status );
    if( status != ISDC_OK ){
      status = RILlogMessage( NULL,Error_2,"Error reading history! status = %d",status );
      return( status );
    }
    if( NumDeadTimeValues != NumDeadTimes ){
      status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
        + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
      return( status );
    }
  }
  else{
    if( chatty > J_CHATTY_NORMAL ){
      status = RILlogMessage( NULL,Warning_2,
                              "No dead time history data found; spectra will not be correcte
d.",
                              status );
      return( status );
    }
  }


  /*
   *
   *  Read in the event data
   *
   */




  /*
   *   Compose selection string
   */

  strcpy(RowFilter, "PI >= " );
  sprintf( strLong,"%d", chanLow );
  strcat(RowFilter, strLong );
  strcat(RowFilter, " && PI <= " );
  sprintf( strLong,"%d", chanHigh );
  strcat(RowFilter, strLong );
  if( chatty > J_CHATTY_NORMAL )
    status = RILlogMessage(NULL,Log_0,
                           "work ##5## RowFilter string: %s",RowFilter);

   /*
   *
   *   Select the events
   *
   */


  NumIntervals = 0;    /*  Signals no selection on OBTime  */
  OBTstart[0] = DAL3_NO_OBTIME;
  OBTend[0] = DAL3_NO_OBTIME;

  status = DAL3JEMXselectEvents(
                                ptr_inSWG,
                                jemxNum,
                                jemxType,
                                COR_NO_OBT,
                                NumIntervals,
                                OBTstart,
                                OBTend,
                                RowFilter,
                                status );

  if(status != ISDC_OK){
    if( DAL3JEMX_NO_JEMX_EVENTS == status ){
      status = RILlogMessage( NULL, Warning_3,
                              "No events of requested type found in SWG." );
      return( ISDC_OK );
    }
    else{
      status = RILlogMessage( NULL, Error_1, "%s status = %d",
                              "DAL3JEMXselectEvents did not like the data!",
                              status );

    }
  }

  /*
   *  Get number of events
   */

  status = DAL3JEMXgetNumEvents( &NumEventsD3J, status );

  if( status != ISDC_OK ) {
    status = RILlogMessage( NULL, Error_1, "%s status = %d",
                            "DAL3JEMXselectEvents did not like the data!",
                            status );
  } else {
    status = RILlogMessage( NULL, Log_0,
                           "work ##8## %ld events have been selected", NumEventsD3J );
    NumEvents = NumEventsD3J;
  }



  /*
   *
   *   Allocate memory for OBT, PI, DETY, DETY and EVT_EVAL event parameter arrays
   *
   */

  status = DALallocateDataBuffer( &buff_timeOBT,NumEvents*sizeof( OBTime ),status );
  if( status != ISDC_OK ){
    status =  RILlogMessage( NULL,Error_3,"Problem assigning memory for OBT",status );
  }
  status = DALallocateDataBuffer( &buff_pi,NumEvents*sizeof( char ),status);
  if( status != ISDC_OK ){
    status =  RILlogMessage( NULL,Error_3, "Problem assigning memory for PI", status );
  }
  status = DALallocateDataBuffer( &buff_detx,NumEvents*sizeof( float ),status);
  if( status != ISDC_OK ){
    status =  RILlogMessage( NULL,Error_3,"Problem assigning memory for DETX",status );
```

```
  }
  status = DALallocateDataBuffer( &buff_dety,NumEvents*sizeof( float ),status);
  if( status != ISDC_OK ){
    status =  RILlogMessage( NULL,Error_3,"Problem assigning memory for DETY",status );
  }
  status = DALallocateDataBuffer( &buff_evteval,NumEvents*sizeof( char ),status);
  if( status != ISDC_OK ){
    status =  RILlogMessage( NULL,Error_3, "Problem assigning memory for EVT_EVAL",status );
  }



  /*
   *  Restructure arrays
   */

  detx = buff_detx;
  dety = buff_dety;
  pi   = buff_pi;
  evteval = buff_evteval;
  timeOBT = buff_timeOBT;







  /* Read OBTs */

  DALtype = DAL3_OBT;
  status = DAL3JEMXgetEvents( OB_TIME, &DALtype, buff_timeOBT, status );
  if( status != ISDC_OK ) return( status );
  if( chatty > J_CHATTY_NORMAL )
    status = RILlogMessage( NULL, Log_0, "Got event parameter: OB_TIME");
  if( NumValues != NumEvents ){
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;

    return( status );
  }

  /* Read PI values */

  DALtype = DAL_BYTE;
  if( jemxType == REST ){
    /*  status = DAL3JEMXgetEvents( REST_PI, &DALtype, buff_pi, status );  */
    status = DALtableGetCol( ptrDS, "REST_PI", 0, &DALtype, &NumValues,
                             buff_pi, status );
    if( status != ISDC_OK ) return( status );
    if( chatty > J_CHATTY_NORMAL )
      status = RILlogMessage(NULL, Log_0,
                             "work ##12## Got event parameter: REST_PI");
    if( NumValues != NumEvents ){
      status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
        + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
      return( status );
    }
  }
  else{
    /*  status = DAL3JEMXgetEvents( PI, &DALtype, buff_pi, status );  */
    status = DALtableGetCol( ptrDS, "PI", 0, &DALtype, &NumValues,
                             buff_pi, status );
    if( status != ISDC_OK ) return( status );
    if( chatty > J_CHATTY_NORMAL )
      status = RILlogMessage(NULL, Log_0,
                               "work ##12## Got event parameter: PI");
    if( NumValues != NumEvents ){
      status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
        + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
      return( status );
    }
  }

  /* Read DETX values */

   DALtype = DAL_FLOAT;
  /*  status = DAL3JEMXgetEvents( DETX, &DALtype, buff_detx, status );  */
  status = DALtableGetCol( ptrDS, "DETX", 0, &DALtype, &NumValues,
                           buff_detx, status );
  if( status != ISDC_OK ) return( status );
  if( chatty > J_CHATTY_NORMAL )
    status = RILlogMessage(NULL, Log_0,
                           "work ##10## Got event parameters: DETX");
  if( NumValues != NumEvents ) {
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
    return( status );
  }

  /* Read DETY values */

  /*  status = DAL3JEMXgetEvents( DETY, &DALtype, buff_dety, status );  */
  status = DALtableGetCol( ptrDS, "DETY", 0, &DALtype, &NumValues,
                           buff_dety, status );
  if( status != ISDC_OK ) return( status );
  if( chatty > J_CHATTY_NORMAL )
    status = RILlogMessage(NULL, Log_0,
                           "work ##11## Got event parameter: DETY");
  if( NumValues != NumEvents ) {
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
    return( status );
  }

  /* Read EVT_EVAL values */

  /*  status = DAL3JEMXgetEvents( EVT_EVAL, &DALtype, buff_evteval, status );  */
  status = DALtableGetCol( ptrDS, "EVT_EVAL", 0, &DALtype, &NumValues,
                           buff_evteval, status );
  if( status != ISDC_OK ) return( status );
  if( chatty > J_CHATTY_NORMAL )
    status = RILlogMessage(NULL, Log_0,
                           "work ##13## Got event parameter: EVT_EVAL");
  if( NumValues != NumEvents ) {
    status = J_ERROR_CODE_START + J_SRCEEXTRACT_ERROR_START
      + J_SLC_ERROR_START + J_SLC_BAD_NUM_VALUES;
    return( status );
  }



  /*   Create mask specification array   */

  /* status = JSrcExtrCreateMask(mask);  */

  /*   Create hole transparency array  */

  /*  status = JSrcExtrCreateHole(chatty,maskElemTransp); */
```

```
  /*   Get Source list  */

  /* numSources = JSrcExtrGetObj(sourcePosX,sourcePosY); */

  /* Calculate effective area and area correlations */

  /*  for (i=0;i<=numSources-1;i=i+1){
    if(chatty >= 100){
      status = RILlogMessage( NULL, Log_0,
                             "(x,y) = %f %f\n", sourcePosX[i],sourcePosY[i]);
      if(status != ISDC_OK) continue;
    }
    status = JSrcEffArea(sourcePosX[i],
                         sourcePosY[i],
                         resSize,
                         mask,
                         &effArea);
    if(status != ISDC_OK) continue;

    if(chatty >= 10){
      status = RILlogMessage( NULL, Log_0,
                             "Eff. Area for source %d = %f\n", i, effArea);
      if(status != ISDC_OK) continue;
    }
    sourceEffArea[i] = effArea;
  }

  status = JSrcEffAreaCor(numSources,sourcePosX, sourcePosY,
                          mask, resSize,
                          sourceEffAreaCor,
                          sourceClosedAreaCor);
  if(status != ISDC_OK) continue;

*/




  /* Allocate memmory for event weights.
     eventWeightsBit[NumEvents] = will store one unsigned long int
     value for each event. Source weights will be
     stored as one bit in the integers.

     numEquations = The number of times that each unique
     value of eventWeightsBit can occur. The dimension
     of weightsBit is the number of linear equations
     that will be solved for source fluxes.*/


  /* Look at old j_src_lc + notes  */



  /*  MODIFY!! Write the event weights to the data element  */



  /*  Calculate source spectra

      status = j_src_spec(); */



  /*  Calculate source light curves

      status = j_src_lc_bin(); */



  /*  De-allocate buffer memory  */

  if( buff_mode != NULL ) { status = DALfreeDataBuffer( buff_mode,status ); }
  if( buff_greyFilter != NULL ) { status = DALfreeDataBuffer( buff_greyFilter,status ); }
  if( buff_greyOBTime != NULL ) { status = DALfreeDataBuffer( buff_greyOBTime,status ); }
  if( buff_deadTime != NULL ) { status = DALfreeDataBuffer( buff_deadTime,status ); }
  if( buff_deadOBTime != NULL ) { status = DALfreeDataBuffer( buff_deadOBTime,status ); }

  if( buff_timeOBT != NULL ) { status = DALfreeDataBuffer( buff_timeOBT,status ); }
  if( buff_pi != NULL ) { status = DALfreeDataBuffer( buff_pi,status ); }
  if( buff_detx != NULL ) { status = DALfreeDataBuffer( buff_detx,status ); }
  if( buff_dety != NULL ) { status = DALfreeDataBuffer( buff_dety,status ); }
  if( buff_evteval != NULL ) { status = DALfreeDataBuffer( buff_evteval,status ); }


  if( status != ISDC_OK ) return( status );

  if( chatty > J_CHATTY_NORMAL ){
    status = RILlogMessage( NULL,Log_0,
                           "work ##26## Buffer memory has been freed",status );
  }

  return( status );
}
```