Please contact OpenAccess Sales
408-350-7040
www.OpenAccessSoftware.com

Application Note for OpenAccess™ SDK

Subject: **ODBC, JDBC, or OLE DB Access to ISAM File Databases**

Date: **Revised March 11, 2003**

Markets: **Vertical applications**

Need

Open Database Connectivity (ODBC) compliant access to data has become a primary requirement by end users on vendors supplying applications containing data. It is becoming even more crucial as corporate organizations accelerate the deployment of client/server computing. Widely used tools such as word processors, spreadsheets and database application development tools are ODBC enabled and they provide means of building powerful end-user applications quickly through ODBC access.

Many vertical applications use ISAM (Indexed Sequential Access Management) type of files for the storage and management of data. These application require all data access to be performed using customized programs and tend to exist on SCO UNIX, VAX/VMS, Alpha/ OpenVMS or Windows NT type of platforms. End users are interested in accessing this information from the desktop for reporting and application. More recently, the ability to make this information available on the web provides a new paradigm for data access and distribution.

In order to make this information easily accessible and usable, one needs a software infrastructure that:

- supports access to information stored in proprietary databases from standard desktop tools like Microsoft Access, PowerBuilder, Visual Basic, Crystal Reports and many other tools. Today this means ODBC and JDBC compatible access.
- allows you to quickly develop client applications for the PC environment using any of the 100s of desktop tools
- can be developed in a very short time without becoming a ODBC, JDBC, OLE DB or SQL expert

Once a user has the ability to use ODBC to access your ISAM database system, the possibilities are endless.

Sample Case

For this application note, assume we have a Widget Management System (WMS) that runs on a UNIX platform. Users access this system from terminals attached to the UNIX box. Currently all interactions with the WMS is by logging into the UNIX box from terminals and running the WMS programs. Only way to transfer data between the WMS and a PC is by exporting a report into a file and then transferring it to a PC. Today, users of the WMS want to get this data directly into Microsoft Excel and Crystal Reports. The WMS stores data using CISAM compatible database.

We want the following features:

1. ODBC/OLE DB access - to provide read/write access to all or some of the data files.
2. Fast data access - optimized access to the files which may contain millions of records.
3. Optimized join processing - handle joins between many files that may contain millions of records.
4. Flexibility in schema management - allow the use of either the existing data dictionary feature in the database or easily set one up using a supplied schema manager.
5. Support for future desktop data access standards - support JDBC, OLE DB and other technologies that may come up in the future without changing the server.
6. Support all my platforms - need server support for Alpha/OpenVMS, VAX/VMS, Intel/NT, Alpha/NT, RS-6000/AIX, Intel/SCO UNIX, HP-9000/HP-UX,

SPARC/Solaris, NCR-3300/AT&T UNIX and IBM MVS. Want to implement code on one platform and easily use on all others.

7. Ease of distribution - we want to be able to easily distribute the server and the client components.

Proposed Solution

The OpenAccess SDK product can help you create a client/server or local ODBC/OLE DB interface to your database within weeks! We provide 90% of the code in binary form. You just have to implement the functions to read and write rows of data from the files in your database. No need to learn ODBC, OLE DB or SQL. Figure 1 shows the proposed architecture using the OpenAccess SDK product. The only portion you develop is the box labeled "Your IP". This code is responsible for accepting a data access request along with the optimization information from the Database Access Manager (DAM) and using this information to build a set of rows.

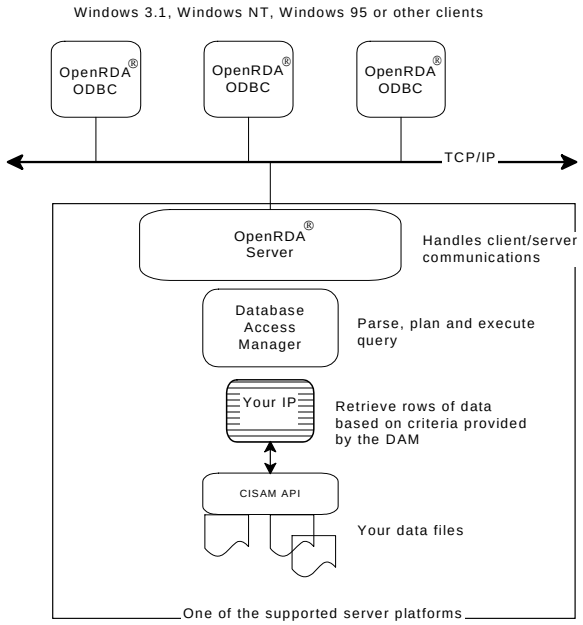

Figure 1: OpenAccess SDK based architecture

Let's walk through the sequence of steps we need to perform in order to execute a SQL statement against the ISAM database. The following steps describe the operation the IP performs in the course of executing a query:

1. Receive connection from the PC client with an user name and password - use this information to provide access verification to your database system. This information can be used to restrict different class users to certain table access. If security does not exist, this information does not have to be used. The IP receives this request when the client issues an ODBC connect call.

2. Receive an execute request - the IP is called to execute a query when the client issues an ODBC SQLExecute call. At this time, the IP can call the functions in the DAM to find out what table is being accessed and what conditions are specified in the query. For example take the query given below.

```
select PART,MFR,DESCRIP from PARTS
where PARTNO = '1FM1'          (Q1)
```

   To execute this query, the IP will first determine the table being accessed and the requested operation. In this case the operation is SELECT and the table is PARTS. Assuming the IP has been setup to use index information, the IP first checks to see if any condition on an indexed column is present. If so, the IP asks the DAM to report restrictions on that column (PART in this example). In this case the DAM will report (=,'1FM1') as a condition on column PART. The IP can find out if additional columns in the query are indexed and then use this information to get restrictions on those columns. The IP then uses all this information to call functions to retrieve the data from the ISAM files. In our example, we use the function `WMS_positionInFile( fileName = 'parts.dbf',indexColumn='PART', value='1FM1')` to position the file pointer to a record where the PART=1FM1. We then call `WMS_readRecord( recBuf )` to read the record at the current position into the recBuf buffer. For each row retrieved from the database file(s), the IP builds a row and passes it to the DAM for further evaluation and for placing in the result set. To build a row, the IP breaks apart the data record in the rcBuf buffer into individual field values and maps them to the columns in the row.

   There is no need to have one table to one file mapping. The IP is effectively exposing views to the client and each of these views can be made up of multiple files. Exposing data from multiple files into one table is sometimes better then requiring the end user to always issue joins. Joins are less efficient to process than single table queries and they complicate the writing of queries.

3. receive a disconnect - the IP frees any resources allocated on behalf of the client.

As seen by the above steps, the OpenAccess SDK makes it very simple for the IP to access data in an optimized

way. The above steps can be implemented in less than 50 lines of code.

Features Suited for ISAM Databases

In this section we want to high light some of the features of the OpenAccess SDK that make it ideal for accessing large or small databases maintained in ISAM type files.

Specifically, the OpenAccess SDK provides:

- access to full details of the query
- optimized join processing
- schema management
- transaction control
- ability to support stored procedure
- ability to implement business rules

Access to Full Details of the Query

The DAM allows the IP to find out as little or as much as it wants about a query. This helps the IP use information contained in the query to limit the number of rows it reads. Information about the query can be obtained on individual columns or on the entire WHERE clause. You can retrieve the entire WHERE clause as a set of expressions on the referenced columns. This information can be used to reformulate the query into your own syntax if your database supports a query language. You can also retrieve the full expression information to implement powerful optimization schemes.

Optimized Join Processing

Joins are very expensive unless handled properly. The DAM handles all joins. The IP is only responsible for accessing data from a single table at a time. For join processing, the IP is called to process a SELECT for each table. The simplest way to implement a join is for the IP to execute the query on one of the tables, execute the query on the next table, and then perform a Cartesian product between the two tables. A Cartesian product of two result sets of size M and N is M*N. This is a very large number as M and N grow and this happens very often when a single piece of information is spread amongst many tables. The DAM optimizes this by first building a result set for the first table and then going through each row in this set and passing the required information to the next query as restrictions. This way the off diagonal elements are not even created. The IP is called to process SELECT on the second table as many times are there are rows in the first result set.

Schema Management

The OpenAccess SDK allows you to provide the schema information (data dictionary) for your database tables by either populating the Schema Database provided as part of the DAM or by implementing schema function in the IP.

Transaction Management

The OpenAccess SDK allows you to perform full transaction control by passing the start transaction and end transaction events to the IP. It is the responsibility of each IP and the database to maintain logs required too implement commit and rollback. Also, we support the SELECT … FOR UPDATE syntax to allow you to implement row level locking of records.

Stored Procedures

OpenAccess SDK allows client applications to invoke stored procedures within the IP. This allows the IP to implement many functions on the server as part of the IP code. A stored procedure is implemented as part of the IP code or within your database system and exposed by a name and the required arguments.

Business Rules

OpenAccess SDK allows the IP to easily implement business logic. Since an IP is exposing a view, it has complete control over how selects, inserts, updates and deletes are processed on each of the tables. The IP can take advantage of this to validate the queries and to guarantee that business rules are enforced. You also can easily check the validity of the input data and control the allowed queries.

Your Development Effort

1. Define the schema for the data to be exposed (2 man days)
2. Implement the IP code to access the tables defined in item 1 – start with our sample (2 man days)
3. Test with ODBC applications (2 man days)
4. Optimize (3 man days)
5. Package up for distribution (2 man days)

Expected time of completion: **12 man days**
Expected time for working prototype: **2 days**

Designing And Coding The IP

NOTE: The full source code for the ISAM IP discussed here is available on our web site at the same location where this application note is . Please contact OpenAccess sales or support if you did not receive it. Follow the instructions in *Installing the Sample ISAM IP* in this application note once you have obtained the required files.

Implementing an IP requires two major tasks: 1) schema management 2) data access management. We will first discuss how the schema needs to be setup to allow reading/writing from ISAM files and then we will discuss the implementation of the IP code that will process the ISAM files to expose the data as per schema definition.

Schema Definition

One of the first things to consider in designing an IP is how the data dictionary will be implemented. OpenAccess requires the IP developer to define the schema for the databases using one of two methods:

1. Populate the built-in schema database
2. Implement schema function in the IP code to handle requests for looking up table, column, index, and other information.

The first method is what our documentation refers to a static schema and is suitable for applications where the schema is not changing because the end users cannot add or modify table definitions. The second method is suited for databases where new tables are being created or existing one being modified by the existing application. For the

In either case, the schema database has to provide the required information to map data from the records in the files to the columns of a table defined for that file. In our example the PARTS table is exposing data in a file parts.dat with an index file parts.idx. The records in the parts.dat file are divided into fields with each field being of a certain size and containing certain type of data. The position of each field and the type of data contained in that field is needed as part of the schema information to allow the IP code to map records from the file into a row required by the SQL engine.

In the schema, for each table we can use the oa_userdata field to supply the associate file name. In our case for the table PARTS we will set oa_userdata to parts.dat. This information will be used by our isopen() function to open the file. For each column definition we can use the oa_userdata to store information about the native data type and position within the record. For example, the definition of column PARTNO would have: 'oa_userdata = 1;4' to indicate that column PARTNO starts at position one in the record and is of type integer.

IP Code

The code that implements the IP needs to be generic in that it uses information stored in the schema database to process the specified queries. This way new tables can be handled without modifying the code.

NOTE: All functions starting with isam_ are part of the ISAM IP and all functions and typedefs starting with DAM or dam are part of the OpenAccess SDK libraries.

The isam_ip_execute() function for this IP would look like the code shown in Listing 1. It is called to process the rows for each table involved in the query execution. First we call dam_describeTable to find out what table is being queried and to obtain the file name that has the data for that table.

Next we find out what all columns are referenced in the query. For each of these columns we obtain the mapping into the file record structure. This is done by the isam_getColumnsInfo function. This function makes use of DAM functions that allow you to walk through either all columns in the table or the columns that are referenced in the query.

Next we call isopen() to open he file associated with the current table being accessed. What we do next depends on if we need to do a full table scan or not. To do this we call call dam_getOptimalIndexAndConditions to check for any conditions on columns that are marked as indexed. We want to pick up cases such as 'where PARTNO=xyz'. In this case the above function call will return a condition list (=,xyz) and we can use this information to jump to the location in the parts.dat file where this matches. Depending on your database design, we can also handle cases such as 'where PARTNO > xyz' by jumping to the record where index value is xyz and then reading all rows whose key values are > xyz.

Assuming the dam_getOptimalIndexAndConditions returns a = condition, we can use isstart() function with the key value to find the first record that matches the key. If the index is marked unique then we are done otherwise we have to keep reading next records until no records matching that key are available. The logic here is to walk through each of the index values and build rows for it. The function isam_buildKeyInfo is called to find out information about the columns that are part of the index. The function isam_buildKeyRec is called to build a key record for the selected condition list. This function uses

the information about the column to file record mapping to build up a key record that can be passed to the isstart() ISAM function. For a query of the form:

```
select * from PARTS where PARTNO=1 or
PARTNO=2
```

we would have two condition lists. First we would process all rows with PARTNO=1 and then all records with PARTNO=2. Once we have moved to the record that matches the key record, we read the record using isread() and then call isam_buildRow to convert the record read from the file into columns of a row that can be passed to the OpenAccess SQL engine. Next, we call dam_isTargetRow to see if the current row matches the full where clause. If it does then we add it to the result set. Now if the index is not defined to be unique, then we need to scan rest of the rows after the current row until we find a non-matching row. This logic can be enhanced to handle BETWEEN, >=, <= and LIKE operators on indexed columns.

If dam_getOptimialIndexAndConditions returns no conditions, then we have to do a full table scan. This happens if the query contains no conditions on columns that are part of an index. In this case we call isstart(ISFIRST) to position to read the first record in the file and continue building and processing the rows until we are at the end.

We have discussed handling select here but implementing update and delete is just a matter of adding extra code right after when dam_isTargetRow is called. If true is returned then you either delete or update the current record. Insert is handled simply by getting the rows to insert from the OpenAccess SQL engine and adding them to your file. Again you would make use of the column to record field mapping to build the record to write into the file.

ISAM Interface

The sample IP explained in this application note makes use of standard ISAM API calls as listed in table 1. In the program listing these functions are highlighted with shading.

Table 1:ISAM API Used in This Example

ISAM Function	Description
isopen	Open the specified file
isclose	Close the specified file
isstart	Position the record pointer to the specified record. A key can be specified. By default the first key for the file is assumed. Key value can be specified.
isread	Read the specified record.

Installing the Sample ISAM IP

Please note that the ISAM IP will not compile and link unless you provide the required ISWRAP.H files and the required library that implements the ISAM API.

1. Obtain source code from http://www.odbcsdk.com/support/oa_samples.asp
2. Create a directory isam under the damip directory
3. Create a directory src under the isam directory and uncompress the source files in it.
4. Now follow the instruction in the IP Examples document you downloaded or received as part of the SDK. In the instructions use ISAM for the IP type and substitute all occurrences of `{example}` with isam. The ISAM IP uses static schema.

What's Next

This application note provides overview and details of using the OpenAccess SDK to implement a ODBC or OLE DB interface to your flavor of ISAM database. You can start with our sample ISAM IP and add:

1. Support for update/delete/insert
2. Cursor based select processing
3. Stored procedures for commonly performed business logic
4. Other data types

Listing 1: Code that implements an IP for CISAM based database

```
/* ISAM_DRV.C
 *
 * (c) OpenAccess Software, Inc.
 *
 * Programer(s):   Dipak Patel
 *
 * Revision:    $Revision: 1.5 $
 *              $Date: 1997/05/01 18:45:20 $
 *              $Source: U:/openrda/dam3_0/dbisamdrv/rcs/isam_drv.c,v $
 *
 * Description: Sample IP to access CISAM files. Has been tested.
 *
 */

static char rcsid[] = "$Id: isam_drv.c,v 1.5 1997/05/01 18:45:20 PRASANNA Exp $";

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "isam_util.h"
#include "isam_drv.h"

static int isam_buildRow( ISAM_STMT_DA * pStmtDA, DAM_HROW hRow );
static int isam_buildKeyRec( ISAM_STMT_DA * pStmtDA, DAM_HCONDLIST hCondList, int * piKeyOp
);
static int isam_buildColumnsInfo( ISAM_STMT_DA * pStmtDA );
static int isam_buildKeyInfo( ISAM_STMT_DA * pStmtDA, DAM_HINDEX hIndex );
static int isam_isValidRec( ISAM_STMT_DA * pStmtDA, DAM_HCONDLIST hCondList);

TM_ModuleCB     isam_tm_Handle;
IP_SUPPORT_ARRAY    isam_support_array =
                    {0,
                    1, /* IP_SUPPORT_SELECT */
                    0, /* IP_SUPPORT_INSERT */
                    0, /* IP_SUPPORT_UPDATE */
                    0, /* IP_SUPPORT_DELETE */
                    0, /* IP_SUPPORT_SCHEMA - IP supports Schema Functions */
                    0, /* IP_SUPPORT_PRIVILEGES  - IP can validate user privileges */
                    1, /* IP_SUPPORT_OP_EQUAL */
                    0, /* IP_SUPPORT_OP_NOT   */
                    0, /* IP_SUPPORT_OP_GREATER */
                    0, /* IP_SUPPORT_OP_SMALLER */
                    0, /* IP_SUPPORT_OP_BETWEEN */
                    0, /* IP_SUPPORT_OP_LIKE    */
                    0, /* IP_SUPPORT_OP_NULL    */
                    0, /* IP_SUPPORT_SELECT_FOR_UPDATE */
                    0, /* IP_SUPPORT_START_QUERY */
                    0, /* IP_SUPPORT_END_QUERY */
                    0, /* IP_SUPPORT_UNION_CONDLIST */
                    0, /* IP_SUPPORT_CREATE_TABLE */
                    0, /* IP_SUPPORT_DROP_TABLE */
                    0, /* IP_SUPPORT_CREATE_INDEX */
                    0, /* IP_SUPPORT_DROP_INDEX */
                    0, /* IP_SUPPORT_PROCEDURE */
                    0, /* IP_SUPPORT_CREATE_VIEW */
                    0, /* IP_SUPPORT_DROP_VIEW */
                    0, /* IP_SUPPORT_QUERY_VIEW */
                    0, /* IP_SUPPORT_CREATE_USER */
                    0, /* IP_SUPPORT_DROP_USER */
                    0, /* IP_SUPPORT_CREATE_ROLE */
                    0, /* IP_SUPPORT_DROP_ROLE */
                    0, /* IP_SUPPORT_GRANT */
                    0, /* IP_SUPPORT_REVOKE */
                    0  /* IP_SUPPORT_PUSHDOWN_QUERY */
```

```
                    };

/*****************************************************************
Function:       isam_ip_init()
Description:    Initialize the Interface Provider and the data source being supported
                and return handle to the context of the driver.

Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
                DAM_NOT_AVAILABLE if Data source is not available or No Driver
*****************************************************************/
int             isam_ip_init(TM_ModuleCB tmHandle, XM_Tree *pMemTree, IP_HENV *phenv)
{
    ISAM_ENV_DA      *pEnvDA;

   /* save the trace module handle */
    isam_tm_Handle = tmHandle;

    tm_trace(isam_tm_Handle, UL_TM_F_TRACE, "isam_ip_init() has been called\n", ());

    /* allocate the environment handle */
    if(!(pEnvDA = (ISAM_ENV_DA *)xm_allocItem(pMemTree,sizeof(ISAM_ENV_DA), XM_NOFLAGS)))
                              return DAM_FAILURE;

    /* save the memory tree handle */
    pEnvDA->pMemTree = pMemTree;

    /* initialize the IP data source */

    /* set the return value */
    *phenv = pEnvDA;

    return DAM_SUCCESS;
}

/*****************************************************************
Function:       isam_ip_exit()
Description:    Shutdown the Interface Provider. Close the data source.
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*****************************************************************/
int             isam_ip_exit(IP_HENV henv)
{
    ISAM_ENV_DA  *pEnv = (ISAM_ENV_DA *)henv;

    tm_trace(isam_tm_Handle, UL_TM_F_TRACE, "isam_ip_exit() has been called\n", ());

    return DAM_SUCCESS;
}


/*****************************************************************
Function:       isam_ip_getInfo()
Description:    Return value for the requested information
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*****************************************************************/
int             isam_ip_getInfo(IP_HENV henv, int iInfoType,
                                void *pInfoValue, int iInfoValueMax, int *piInfoValueLen)
{
    ISAM_ENV_DA  *pEnv = (ISAM_ENV_DA *)henv;

    return DAM_SUCCESS;
}

/*****************************************************************
Function:       isam_ip_getSupport()
```

```
Description:    Return value for the requested information
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
**********************************************************************/
int             isam_ip_getSupport(IP_HENV henv, int iSupportType,
                                   int *pSupportExists)
{
    ISAM_ENV_DA  *pEnv = (ISAM_ENV_DA *)henv;

    *pSupportExists = isam_support_array[iSupportType];

    return DAM_SUCCESS;
}

/**********************************************************************
Function:       isam_ip_connect()
Description:    Connect to the data source being specified using the
                authentication information
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
**********************************************************************/
int             isam_ip_connect(DAM_HDBC dam_hdbc, IP_HENV henv,
                                XM_Tree *pMemTree, char *sDbName, char *sUserName, char
*sPassword,
                                IP_HDBC *phdbc)
{
    ISAM_ENV_DA      *pEnvDA = (ISAM_ENV_DA *)henv;
    ISAM_CONN_DA     *pConnDA;

    tm_trace(isam_tm_Handle, UL_TM_F_TRACE, "isam_ip_connect() has been called\n", ());

    /* allocate the connection da */
    if(!(pConnDA = (ISAM_CONN_DA *)xm_allocItem(pMemTree,sizeof(ISAM_CONN_DA), XM_NOFLAGS)))
        return DAM_FAILURE;

    /* initailze the ConnDA */
    pConnDA->pMemTree = pMemTree;
    pConnDA->pEnvDA = pEnvDA;

    /* connect to the given data source */

    /* set the return value */
    *phdbc = pConnDA;

    return DAM_SUCCESS;
}


/**********************************************************************
Function:       isam_ip_disconnect()
Description:    Disconnect from the data source. Close open file handles etc.
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
**********************************************************************/
int             isam_ip_disconnect(DAM_HDBC dam_hdbc, IP_HDBC hdbc)
{
    ISAM_CONN_DA     *pConnDA = (ISAM_CONN_DA *)hdbc;

    tm_trace(isam_tm_Handle, UL_TM_F_TRACE, "isam_ip_disconnect() has been called\n", ());

    /* disconnect from the data source */

    /* free the connection da */
    xm_freeItem(pConnDA);

    return DAM_SUCCESS;
}
```

```
/************************************************************************
Function:       isam_ip_startTransaction()
Description:    Start a new transaction on the connection
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
************************************************************************/
int             isam_ip_startTransaction(DAM_HDBC dam_hdbc, IP_HDBC hdbc)
{
    ISAM_CONN_DA     *pConnDA = (ISAM_CONN_DA *)hdbc;

    /* start a new transaction */

    return DAM_SUCCESS;
}

/************************************************************************
Function:       isam_ip_endTransaction()
Description:    End the transaction on the connection. Use the iType to
                either commit or rollback
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
************************************************************************/
int             isam_ip_endTransaction(DAM_HDBC dam_hdbc, IP_HDBC hdbc, int iType)
{
    ISAM_CONN_DA     *pConnDA = (ISAM_CONN_DA *)hdbc;

    /* end the transaction */
    if (iType == DAM_COMMIT) {
        }
    else if (iType == DAM_ROLLBACK) {
        }

    return DAM_SUCCESS;
}


/************************************************************************
Function:       isam_ip_execute()
Description:    Execute the given statement
Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
************************************************************************/
int             isam_ip_execute(IP_HDBC hdbc,
                                DAM_HSTMT hstmt,
                                int iStmtType,
                                DAM_HCOL hSearchCol,
                                int *piNumResRows)
{
    ISAM_CONN_DA     *pConnDA = (ISAM_CONN_DA *)hdbc;
    ISAM_STMT_DA     *pStmtDA;
    XM_Tree         *pMemTree;
    DAM_HROW        hRow;
    int             rc, iBufLen;
    int             iKeyOp;
    DAM_HINDEX      hIndex;
    DAM_HSET_OF_CONDLIST hSetOfCondList;

    tm_trace(isam_tm_Handle, UL_TM_F_TRACE, "isam_ip_execute() has been called. There is %s
             search column\n", (hSearchCol ? "a" : "NO"));

    /* get the memory tree to be used */
    pMemTree = dam_getMemTree(hstmt);

    /* allocate a new stmt */
    if(!(pStmtDA = (ISAM_STMT_DA *)xm_allocItem(pMemTree,sizeof(ISAM_STMT_DA), XM_NOFLAGS)))
        return DAM_FAILURE;
```

```
    /* initialize the StmtDA */
    pStmtDA->pMemTree = pMemTree;
    pStmtDA->pConnDA = pConnDA;
    pStmtDA->dam_hstmt = hstmt;
    pStmtDA->iType = iStmtType;
    pStmtDA->buf = NULL;
    pStmtDA->pKeyBuf = NULL;

    /* get the table information */
     dam_describeTable(pStmtDA->dam_hstmt, NULL, NULL, pStmtDA->sTableName,
                    pStmtDA->sFileName, pStmtDA->sUserdata);

    /* initialize the result */
    *piNumResRows = 0;

    /* build column to record field mapping array and allocate buffer space*/
    isam_buildColumnsInfo( pStmtDA );

    iBufLen = atoi(pStmtDA->sUserdata);
    pStmtDA->buf = (char *)xm_allocItem(pMemTree,iBufLen,XM_NOFLAGS);

    /* open the table */
    rc = isopen( pStmtDA->sFileName , ISINPUT + ISRDONLY );
    if ( rc < SUCCESS ){
    tm_trace(isam_tm_Handle, UL_TM_ERRORS, "isam_open_table(): error %d on isopen",
                            (*is_errno( rc )));
        return DAM_FAILURE;
    }else{
        pStmtDA->fd = rc;
    }

    /* Determine if the query contains conditions on Indexed columns */
    dam_getOptimalIndexAndConditions( pStmtDA->dam_hstmt, &hIndex, &hSetOfCondList);
    if( hIndex ){
        /* A index has been selected that can be used to optimize access for this query. The
           index can be a single column or multiple column index. We need to obtain the key
           part for each column and add it to the key record structure. To do this, first
           we need to find out about each of the columns for which we are specifying the key.
        */

        DAM_HCONDLIST   hCondList;

        pStmtDA->pKeyBuf = (char *)xm_allocItem(pMemTree,iBufLen,XM_NOFLAGS);
        isam_buildKeyInfo( pStmtDA, hIndex );

        /* now we need to build rows for each condition list */
        hCondList = dam_getFirstCondList( hSetOfCondList );
        while( hCondList != NULL ){
            isam_buildKeyRec( pStmtDA, hCondList,&iKeyOp);
            /* Now build row for this key record */
            rc = isstart(pStmtDA->fd , NULL, 0, pStmtDA->pKeyBuf, ISEQUAL);
            if( rc == 0 ){ /* found a matching row */
                if (!isread( pStmtDA->fd , pStmtDA->buf, ISNEXT )){
                    hRow = dam_allocRow( pStmtDA->dam_hstmt );
                    rc = isam_buildRow( pStmtDA, hRow);
                    if( rc != DAM_SUCCESS)
                        return DAM_FAILURE;
                    if (dam_isTargetRow( pStmtDA->dam_hstmt,hRow) == DAM_TRUE)
                        dam_addRowToTable(pStmtDA->dam_hstmt, hRow);
                    else
                        dam_freeRow(hRow);
                    *piNumResRows++;
                }

                /* Now handle the remaining rows that may still match this condition. This
will
```

```
                   happen when the key is non-unique*/
                if( pStmtDA->iIndexType == 1){
                    while (!isread( pStmtDA->fd , pStmtDA->buf, ISNEXT )){
                        if( isam_isValidRec(pStmtDA, hCondList) == FALSE )
                            break;
                        hRow = dam_allocRow( pStmtDA->dam_hstmt );
                        rc = isam_buildRow( pStmtDA, hRow);
                        if( rc != DAM_SUCCESS)
                            return DAM_FAILURE;
                        /* check to see if the key part of the row matches the criteria */
                        if (dam_isTargetRow( pStmtDA->dam_hstmt,hRow) == DAM_TRUE)
                            dam_addRowToTable(pStmtDA->dam_hstmt, hRow);
                        else
                            dam_freeRow(hRow);
                        *piNumResRows++;
                    }
                }
            }
            hCondList = dam_getNextCondList( hSetOfCondList);
        }

    }else{

        /* No restrictions on indexed columns so we need to do a table scan */

        rc = isstart(pStmtDA->fd , NULL, 0, NULL, ISFIRST);
        if ( rc < SUCCESS ){
            tm_trace(isam_tm_Handle, UL_TM_ERRORS, "isam_ip_execute(): error %d on isstart",
                                                    ( *is_errno( rc )));
            return DAM_FAILURE;
        }

        while( !isread( pStmtDA->fd , pStmtDA->buf, ISNEXT ) ){

            /* Map the row into columns that are part of the query */
            hRow = dam_allocRow( pStmtDA->dam_hstmt );
            rc = isam_buildRow( pStmtDA, hRow);
            if( rc != DAM_SUCCESS)
                return DAM_FAILURE;

            if (dam_isTargetRow( pStmtDA->dam_hstmt,hRow) == DAM_TRUE)
                dam_addRowToTable(pStmtDA->dam_hstmt, hRow);
            else
                dam_freeRow(hRow);

            *piNumResRows++;
        }
    }

    /* close the table */
    rc = isclose( pStmtDA->fd );
    if ( rc < SUCCESS ){
       tm_trace(isam_tm_Handle, UL_TM_ERRORS, "isam_ip_execute: error %d on isclose",
       (*is_errno( rc )));
        return DAM_FAILURE;
    }else{
        pStmtDA->fd = -1;
    }

    /* free memory allocated for the statement */
    if( pStmtDA->buf )
        xm_freeItem( pStmtDA->buf);
    if( pStmtDA->pKeyBuf )
        xm_freeItem( pStmtDA->pKeyBuf);

    if( pStmtDA->pColInfo )
        xm_freeItem( pStmtDA->pColInfo);
```

```
    xm_freeItem( pStmtDA );

    tm_trace(isam_tm_Handle, UL_TM_INFO, "isam_ip_execute() Query executed successfully\n",
());

    return DAM_SUCCESS;
}


/*********************************************************************
Function:       isam_buildColumnInfo()
Description:    Walk through all the columns in the query and obtain the
                information required to perform the file record to ROW
                mapping.
                extract the starting and end position from the szUserData. We assume it will
have the
                following format: recstart,recend
                recend is not used for integer and other basic types as the number of bytes
                required to store those types are fixed.

Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*********************************************************************/
int isam_buildColumnsInfo( ISAM_STMT_DA * pStmtDA )
{
    DAM_HCOL    hCol;
    int         hColCnt = 0, i=0;
    char        szUserData[255];
    char        szColName[DAM_MAX_ID_LEN+1];

    hCol = dam_getFirstCol(pStmtDA->dam_hstmt, DAM_COL_IN_USE);
    while ( hCol != NULL){
        hColCnt++;
        hCol = dam_getNextCol(pStmtDA->dam_hstmt);
    }
    pStmtDA->pColInfo = (ISAM_COL_INFO *)xm_allocItem(pStmtDA->pMemTree,
                        sizeof( ISAM_COL_INFO)*hColCnt,XM_NOFLAGS);

   /* now allocate space to store the additiona information */
    hCol = dam_getFirstCol(pStmtDA->dam_hstmt, DAM_COL_IN_USE);
    while ( hCol != NULL){
        dam_describeColDetail(hCol, NULL, NULL,NULL,NULL, NULL, NULL, szUserData);
        dam_describeCol( hCol, NULL, szColName, &pStmtDA->pColInfo[i].iColType,NULL);
        sscanf(szUserData,"%d,%d,%d",&pStmtDA->pColInfo[i].iStart,&pStmtDA->pColInfo[i].iEnd,
&pStmtDA->pColInfo[i].iScale);
        tm_trace(isam_tm_Handle, UL_TM_MAJOR_EV, "isam_ip_execute(): Column Info: Name:%s
Start: %d End: %d", (szColName, pStmtDA->pColInfo[i].iStart,pStmtDA->pColInfo[i].iEnd));
        hCol = dam_getNextCol(pStmtDA->dam_hstmt);
        i++;
    }

    return DAM_SUCCESS;

}

/*********************************************************************
Function:       isam_buildRow()
Description:    Takes an empty HROW as in input and fills it with data from
                a record that has been read in memory. It uses the column to
                record field mapping information contained in the OA_USERDATA
                field of each column defintion.


Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*********************************************************************/
```

```
int isam_buildRow( ISAM_STMT_DA * pStmtDA, DAM_HROW hRow )
{
    DAM_HCOL        hCol;
    int             rc;
    int             i=0, iStart, iEnd, iScale;

    hCol = dam_getFirstCol(pStmtDA->dam_hstmt, DAM_COL_IN_USE);
    while ( hCol != NULL){
        iStart = pStmtDA->pColInfo[i].iStart;
        iEnd   = pStmtDA->pColInfo[i].iEnd;
        iScale  = pStmtDA->pColInfo[i].iScale;
        switch (pStmtDA->pColInfo[i].iColType){
        case XO_TYPE_CHAR:
            dam_addValToRow( pStmtDA->dam_hstmt,hRow, hCol, XO_TYPE_CHAR, pStmtDA->buf +
                        (iStart-1),(iEnd-iStart+1));
            break;

        case XO_TYPE_NUMERIC:
            {
                char * pBuf = pStmtDA->buf + iEnd-1;
                char sNumeric[32];
                char * pNumeric = &sNumeric[31];
                unsigned char iLsb, iMsb;

                *pNumeric-- = '\0';

                while(iStart <= iEnd ){
                    iLsb = *pBuf & 0x0f;

                    if( iLsb <= 9 ){
                        if ( iScale==0)
                            *pNumeric-- = '.';
                        *pNumeric-- = iLsb + '0';
                        iScale--;
                    }

                    iMsb = ( *pBuf-- >> 4 ) & 0xf;

                    if( iMsb <= 9 ){
                        if ( iScale==0)
                            *pNumeric-- = '.';
                        *pNumeric-- = iMsb + '0';
                        iScale--;
                    }

                    iStart ++;
                }
                dam_addValToRow( pStmtDA->dam_hstmt,hRow, hCol, XO_TYPE_CHAR,pNumeric+1,
                                XO_NTS);
                break;
            }

        }

        hCol = dam_getNextCol(pStmtDA->dam_hstmt);
        i++;
    }

    return DAM_SUCCESS;
}

/*********************************************************************
Function:       isam_buildKeyInfo()
Description:    Walk through all the columns referenced by the selected
                index and store the information.

Return:         DAM_SUCCESS on Success
```

```
                DAM_FAILURE   on Failure
*********************************************************************/
int isam_buildKeyInfo( ISAM_STMT_DA * pStmtDA, DAM_HINDEX hIndex )
{
    int             iIndexCnt = 0;
    char            szColName[DAM_MAX_ID_LEN+1];
    char            szUserData[255];
    DAM_HINDEX_COL  hIndexCol;
    DAM_HCOL        hCol;

    dam_describeIndex( hIndex,NULL, pStmtDA->szIndexName,NULL, &pStmtDA->iIndexType,
&pStmtDA->iIndexLen);
    pStmtDA->pIndexColInfo = (ISAM_COL_INFO *)xm_allocItem(pStmtDA->pMemTree, sizeof(
ISAM_COL_INFO)*pStmtDA->iIndexLen,XM_NOFLAGS);

    hIndexCol = dam_getFirstIndexCol( hIndex);
    while ( hIndexCol != NULL ){
        dam_describeIndexCol( hIndexCol, NULL, szColName, NULL);  /* get info about column */
        hCol = dam_getCol( pStmtDA->dam_hstmt, szColName );  /* convert col name to handle */
        dam_describeColDetail(hCol, NULL, NULL,NULL,NULL, NULL, NULL, szUserData);
        sscanf(szUserData,"%d,%d,%d",&pStmtDA->pIndexColInfo[iIndexCnt].iStart,&pStmtDA-
>pIndexColInfo[iIndexCnt].iEnd, &pStmtDA->pIndexColInfo[iIndexCnt].iScale);
        iIndexCnt ++;
        hIndexCol = dam_getNextIndexCol( hIndex );
    }

    return DAM_SUCCESS;

 }


/*********************************************************************
Function:       isam_buildKeyRec()
Description:    Reads the conditions on the selected index columns and builds a key record
                that can be used by ISAM to locate the matching data record.


Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*********************************************************************/
int isam_buildKeyRec( ISAM_STMT_DA * pStmtDA, DAM_HCONDLIST hCondList, int * piKeyOp  )
{

        int             iIndexCnt = 0;
        DAM_HCOND       hCond;

        int             iLeftOpType, iRightOpType, iLeftXoType, iRightXoType;
        int             iIndexOp;

        void            * pLeftData, * pRightData;
        int iKeyPart=0;

        /* A index has been selected that can be used to optimize access for this query. The
           index can be a single column or multiple column index. We need to obtain the key
           part for each column and add it to the key record structure.
        */

        /* get the parts of the key and build a key record */
        hCond = dam_getFirstCond( pStmtDA->dam_hstmt, hCondList );
        while( hCond != NULL ){
            dam_describeCond( hCond, &iLeftOpType, &pLeftData, &iLeftXoType,
                                     &iRightOpType,&pRightData, &iRightXoType);
            switch( iLeftOpType ){
            case SQL_OP_EQUAL:
                iIndexOp = SQL_OP_EQUAL;
                switch( iLeftXoType ){
```

```
                case XO_TYPE_CHAR:
                    {
                        char * pKeyBuf = pStmtDA->pKeyBuf + pStmtDA-
>pIndexColInfo[iKeyPart].iStart - 1;
                        int iLen = pStmtDA->pIndexColInfo[iKeyPart].iEnd - pStmtDA-
>pIndexColInfo[iKeyPart].iStart + 1;
                        memset( pKeyBuf, ' ', iLen); /* pad it */
                        memcpy( pKeyBuf, pLeftData, iLen);
                    }
                    break;
                }
               break;
            }
            hCond = dam_getNextCond(pStmtDA->dam_hstmt, hCondList);
            iKeyPart ++;
        }

        *piKeyOp = iIndexOp;

        return DAM_SUCCESS;
}


/*********************************************************************
Function:       isam_isValidRec()
Description:    Compare the record read from the file with the condition
                in the query to make sure it is part of the valid selection.
                This is to know when to stop reading records once the record
                matching the given index has been located. This code would not
                be needed if the uderlying ISAM database supported moving to
                next row with a matching key.

Return:         DAM_SUCCESS on Success
                DAM_FAILURE   on Failure
*********************************************************************/
int isam_isValidRec( ISAM_STMT_DA * pStmtDA, DAM_HCONDLIST hCondList)
{

        int        iIndexCnt = 0;
        DAM_HCOND  hCond;

        int        iLeftOpType, iRightOpType, iLeftXoType, iRightXoType;
        int        iIndexOp;

        void       * pLeftData, * pRightData;
        int        iMatch = TRUE;
        int iKeyPart=0;

        /* get the parts of the key and validate against current record*/
        hCond = dam_getFirstCond( pStmtDA->dam_hstmt, hCondList );
        while( hCond != NULL && iMatch ){
            dam_describeCond( hCond, &iLeftOpType, &pLeftData, &iLeftXoType,
                                     &iRightOpType,&pRightData, &iRightXoType);
            switch( iLeftOpType ){
            case SQL_OP_EQUAL:
                iIndexOp = SQL_OP_EQUAL;
                switch( iLeftXoType ){

                case XO_TYPE_CHAR:
                    {
                        char * pBuf = pStmtDA->buf + pStmtDA->pIndexColInfo[iKeyPart].iStart
- 1;
                        int iLen = pStmtDA->pIndexColInfo[iKeyPart].iEnd - pStmtDA-
>pIndexColInfo[iKeyPart].iStart + 1;
                        if (!dam_evaluateColCond(pStmtDA->dam_hstmt, hCond, iLeftXoType,
pBuf, iLen ))
                            iMatch = FALSE;
```

```
                    }
                    break;
                }
               break;
            }
            hCond = dam_getNextCond(pStmtDA->dam_hstmt, hCondList);
            iKeyPart ++;
        }

        return iMatch;
}


/* ISAM_UTIL.H
 *
 * (c) OpenAccess Software, Inc.
 *
 * Programer(s):   Dipak Patel
 *
 * Revision:    $Revision: 1.1 $
 *              $Date: 1996/06/24 17:01:38 $
 *              $Source: U:/openrda/dam3_0/dbisamdrv/rcs/isam_util.h,v $
 *
 *
 * Description:     This header file contains IP specific definitons
 *
 */

#ifndef __ISAMUTIL_H
#define __ISAMUTIL_H

#include "ipdef.h"
#include "iswrap.h"

/* Environment Descriptor Area */
typedef struct isam_env_struct {void *pMemTree;/* Memory Tree for the global context
*/
    } ISAM_ENV_DA;

/* Connection Descriptor Area */
typedef struct isam_connection_struct {
    XM_Tree     *pMemTree;
    ISAM_ENV_DA  *pEnvDA;        /* handle to EnvDA */
    } ISAM_CONN_DA;

typedef struct isam_col_info_struct{
    int iStart;
    int iEnd;
    int iScale;
    int iColType;
} ISAM_COL_INFO;

/* Statement Descriptor Area */
typedef struct isam_statement_struct {
    ISAM_CONN_DA *pConnDA;       /* handle to connection descriptor area */
    XM_Tree     *pMemTree;
    DAM_HSTMT   dam_hstmt;      /* DAM handle to the statement */
    int         iType;          /* Type of the query */
    char        sTableName[DAM_MAX_ID_LEN+1];   /* Name of the table being queried
*/
    char        sFileName[255];
```

```
    char        sUserdata[255];
    int         fd;
    char        * buf;
    char        * pKeyBuf;
    ISAM_COL_INFO * pColInfo;
    ISAM_COL_INFO * pIndexColInfo;
    char        szIndexName[DAM_MAX_ID_LEN+1];
    int         iIndexLen, iIndexType;

    } ISAM_STMT_DA;

extern TM_ModuleCB     isam_tm_Handle; /* declared in isam_drv.c */

#endif  /* __ISAMUTIL_H */
```