

HDF5-SRB Model User's Guide

Peter Cao, NCSA
Mike Wan, SDSC

September, 2005

NCSA
1205 W. Clark St
Urbana, IL 61801
Phone: 217-244-0072

SDSC
9500 Gilman Drive, MC 0505
La Jolla, CA 92093-0505
Phone :858-534-5000

Table of Contents

1. Introduction	5
1.1 Overview of SRB	5
What is SRB	6
SRB architecture	6
1.2 Overview of HDF5	7
What is HDF	7
HDF5 file structure.....	8
2. The HDF-SRB Model	9
2.1 Design requirements	9
2.2 The HDF-SRB architecture	9
2.3 The data objects.....	10
H5File	11
H5Group	11
H5Dataset.....	12
2.4 The client and server APIs	13
Pack/Unpack routine enhancements.....	13
General proxy function	14
3 Client and Server Applications	17
3.1 The source code	17
3.2 Configuring and compiling the SRB server.....	18
Configure and build HDFSRB from scratch.....	18
Configure and build HDFSRB from installed SRB	18
3.3 Configuring, writing and compiling the client code	19
test/Makefile	19
Client header files	19
Exmample of .MdasEnv.....	19
Example of . MdasAuth	20
3.4 HDFView – the Java client application	20
The HDF-SRB JNI	20
The Java HDF-SRB objects	21
The GUI components	22

1. Introduction

This document provides information about the HDF-SRB system, and instructions on how to use it.

The HDF-SRB model is a client-server model that provides interactive and efficient access to remote HDF5 files. Like most client/server model, the HDF-SRB model is implemented on a set of client and server APIs and message passing scheme. Unlike other client/server model, the HDF-SRB model is object oriented. The client can access datasets or subsets of datasets in large files without bringing entire files into local machines.

Storing massive data presents two big challenges: management of distributed data systems and efficient access to complex data types. The NCSA Hierarchical Data Format (HDF) and the SDSC Storage Resource Broker (SRB) have addressed the two issues. The SRB is client-server middleware (or grid data software) that provides a uniform interface and authorization mechanism to access heterogeneous data resources (UNIX FS, HPSS, UniTree, DBMS, etc.) distributed on multiple hosts and diverse platforms. The HDF is a file format and software library for storing all kinds of data, simple integers and floats or complex users defined compound data types. The HDF employs a common data model with standard library APIs, providing efficient data storage and I/O access.

The HDF and the SRB offer valuable and complementary data management services, but they have not previously been integrated in an effective way. Earlier work had the SRB accessing HDF data either (a) by extracting entire HDF files, or (b) by extracting byte-streams through the SRB's POSIX interface. Approach (a) fails to take advantage of HDF's ability to offer interactive and efficient access to complex collections of objects. Approach (b) has been shown to be far too low-level to perform reasonably for some data extraction operations.

In discussions between NCSA and SDSC, it has been determined that a more effective approach is possible, one that uses modified HDF APIs on the server side to extract data from large files at the instruction of client-side HDF APIs and SRB as middleware to transfer data between the server and client. This approach would insert the HDF library and other object-level HDF-based libraries (such as HDF-EOS) between the SRB and a data storage source (such as a file system), making it possible to extract objects, rather than files or byte streams. Furthermore, these libraries typically offer query, subsetting, sub-sampling, and other object-level operations, so that these services might also be available.

The HDF-SRB was funded and Sponsored by The National Laboratory for Advanced Data Research (NLADR), The National Science Foundation (NFS) Partnerships for Advanced Computational Infrastructure (PACI) Project in Support of an NCSA-SDSC Collaboration. The first phase of the project was to provide a working prototype of the HDF-SRB system. For more information about the project, please read the project report at <http://hdf.ncsa.uiuc.edu/hdf-srb-html/HDF-SRB-project-report.html>

1.1 Overview of SRB

This section gives a overview of SRB. For details, visit the SRB website at <http://www.sdsc.edu/srb/>.

What is SRB

The SDSC Storage Resource Broker (SRB) is client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. SRB, in conjunction with the Metadata Catalog (MCAT), provides a way to access data sets and resources based on their attributes and/or logical names rather than their names or physical locations. SRB provides:

- ✓ Distributed data storage in multiple types of storage resources across multiple hosts and OS platforms.
- ✓ A common user interface for transparent access to multiple types of
- ✓ storage resources across multiple hosts and OS platforms.
- ✓ A global logical name space to make it easy for users to organize their data.
- ✓ A global user name space to make it easy for system administrators to manage user accounts across multiple hosts and to allow users to use a single sign on to access all resources in the system.
- ✓ A fine-grained access control mechanism.
- ✓ A metadata management system which can be used to associate user-defined metadata with data objects. A well defined metadata querying interface is also provided.

The SRB has been used to implement data grids for data sharing across multiple resources, digital libraries (to support collection-based management of distributed data), and persistent archives (to manage technology evolution). The SRB is in widespread use, supporting collections that have up to 25 million data objects.

SRB architecture

The Storage Resource Broker (SRB) is a middleware that provides distributed clients with uniform access to diverse storage resources in a heterogeneous computing Environment.

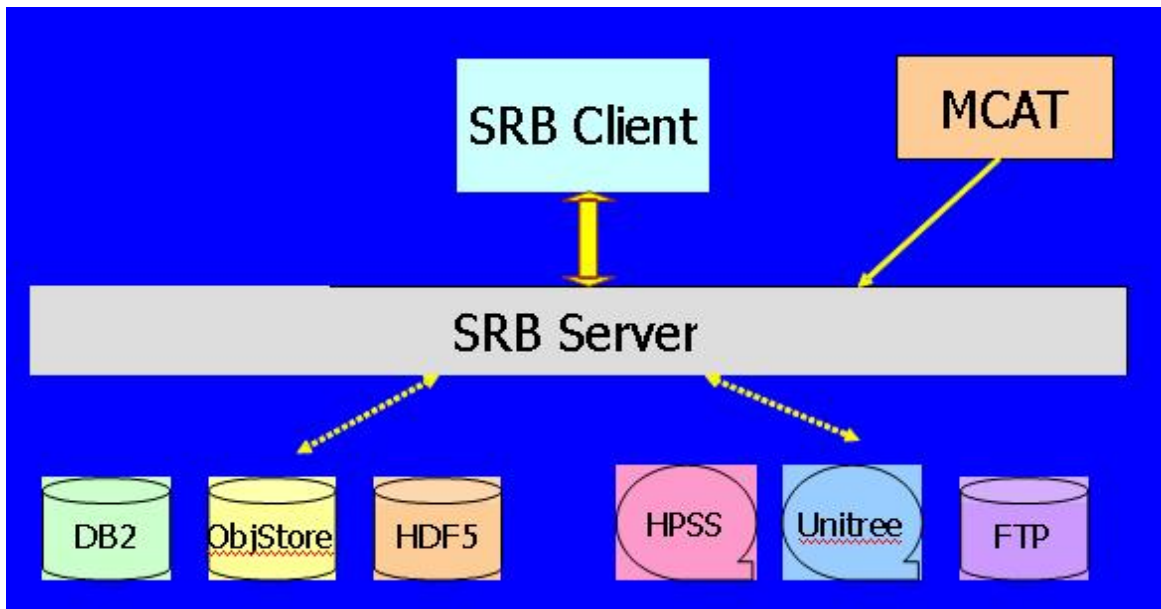


Figure 1 -- A simplified view of the SRB middleware

Figure 1 gives a simplified view of the SRB architecture. The model consists of three components: the meta data catalog (MCAT) service, SRB servers and SRB clients, connected to each other via network.

The MCAT stores meta data associated with data sets, users and resources managed by the SRB. The MCAT server handles requests from the SRB servers. These requests include information queries as well as instructions for meta data creation and update.

Client applications are provided with a set of API for sending requests and receiving response to/from the SRB servers. The SRB server is responsible for carrying out tasks to satisfy the client requests. These tasks include interacting with the MCAT service, and performing I/O on behalf of the clients. A client uses the same common API to access every storage systems managed by the SRB. The complex tasks of interacting with various types of storage system and OS/hardware architecture, are handled by the SRB server.

1.2 Overview of HDF5

This section gives a brief overview of HDF5. For more details, visit the HDF website at <http://hdf.ncsa.uiuc.edu/>

What is HDF

The Hierarchical Data Format (HDF) is a file format for storing and managing scientific data. There are two basic versions of HDF: HDF4 and HDF5. HDF4 is the earlier version and HDF5 is the new version. The two versions are incompatible. HDF4 has limit of 20,000 number of objects and 2 gigabytes in file size. The HDF5 has more improved features and performance. The HDF-SRB system supports HDF5 only.

- ✓ **HDF5 is a file format for storing all kinds of data.** HDF5 can store two primary objects: datasets and groups. A dataset is essentially a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file. Using these two basic objects, one can create and store almost any kind of scientific data structure, such as images, arrays of vectors, and structured and unstructured grids. You can also mix and match them in HDF5 files according to your needs
- ✓ **HDF5 provides efficient data storage and I/O access.** HDF5 was created to address the data management needs of scientists and engineers working in high performance, data intensive computing environments. As a result, the HDF5 library and format emphasize storage and I/O efficiency. For instance, the HDF5 format can accommodate data in a variety of ways, such as compressed or chunked. And the library is tuned and adapted to read and write data efficiently on parallel computing systems.
- ✓ **HDF5 is a library with standard APIs.** Data can be stored in HDF5 in an endless variety of ways, so it is important for communities of users to standardize on how their data is to be organized in HDF5. This makes it possible to share data easily, and also to build and share tools for accessing and analyzing data stored in HDF5. The NCSA HDF team works with users to encourage them to organize HDF5 files in standard ways.
- ✓ **HDF5 provides software and tools.** NCSA maintains a suite of free, open source software, including the HDF5 I/O library and several utilities. The HDF5 user community also develops and contributes software, much of it freely available. Unlike HDF4, there is little commercial support for HDF5 at this time, but we are successfully working with vendors to change this.

HDF5 file structure

HDF5 files are organized in a hierarchical structure, with two primary structures: groups and datasets.

- ✓ **HDF5 group:** a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata.
- ✓ **HDF5 dataset:** a multidimensional array of data elements, together with supporting metadata.

Working with groups and group members is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, objects in an HDF5 file are often described by giving their full (or absolute) path names. For example,

/ -- signifies the root group.

/foo -- signifies a member of the root group called foo.

/foo/zoo -- signifies a member of the group foo, which in turn is a member of the root group.

Any HDF5 group or dataset may have an associated attribute list. HDF5 attributes are small named datasets that are attached to primary datasets, groups, or named datatypes. Attributes can be used to describe the nature and/or the intended usage of a dataset or group.

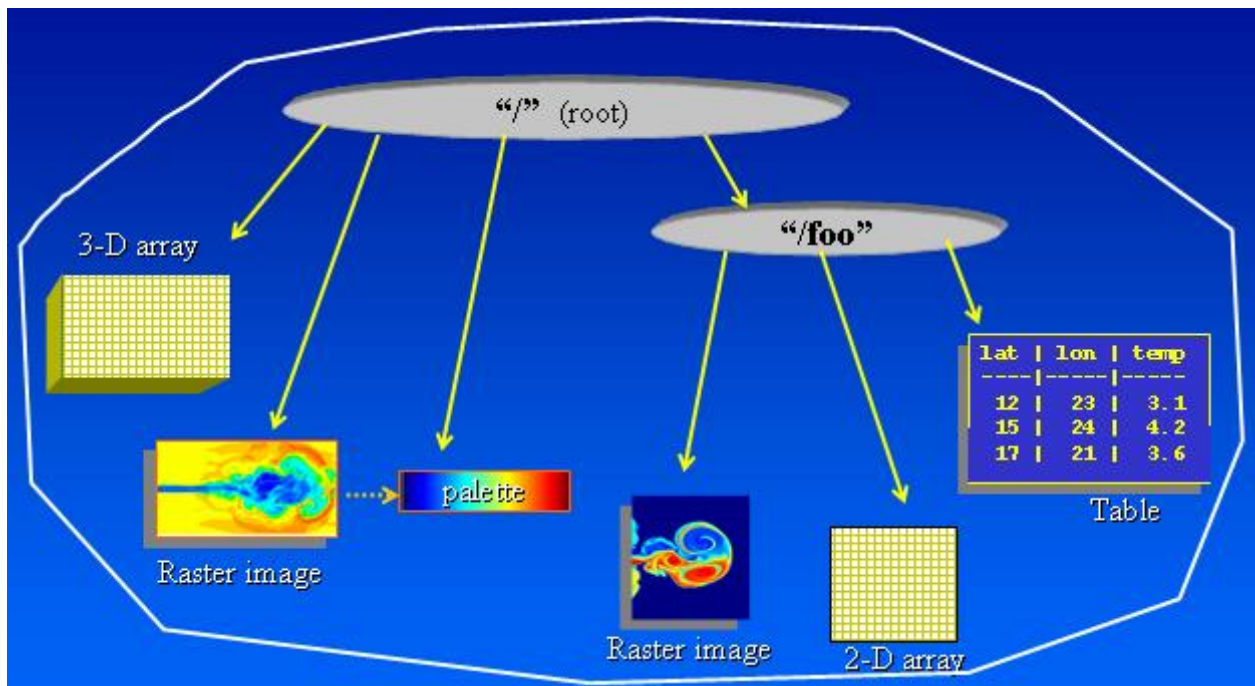


Figure 2 -- Structure of an example HDF5 file

2. The HDF-SRB Model

The HDF and SRB technologies deal with data management in different aspect. The HDF emphasis on efficient data access and complicate data operations on file and SRB focus on data distribution and storage. The goal of the HDF-SRB model is to bring these two technologies together to create client/server system with efficient data access.

The HDF-SRB model is implemented on a set of standard client/server APIs and data structures. The data structures represent the HDF5 objects such as groups, datasets and attributes. Information of client requests and server results are encapsulated in the data structures and packed into SRB message. A set of client/server APIs is defined and used to transfer the SRB message between the client and server.

2.1 Design requirements

The following is a list of design requirements applied to the HDF-SRB model.

- ✓ The HDF-SRB model should be built on object oriented fashion. Using data objects (or structures) has multi-fold benefits.
 - The current HDF and SRB development is independent of the HDF-SRB model. No change is needed to the current HDF and SRB. The HDF-SRB model is added to SRB as new proxy. It does not affect other part of the SRB project. Also, there is no change in the HDF library. The HDF-SRB model is built on a high-level data objects (or structures).
 - It is much easier to support complicated data access such as hyperslab selection in HDF5. Since information of client request and server results are encapsulated in the data objects and the data objects are self explained, the client can pass any user requests to the server.
 - Easy to write client application based on a standard set of client stubs.
- ✓ The HDF-SRB model must be simple
 - Use one client API and one server API
 - No complicated packMsg()/unpackMsg()
- ✓ The HDF-SRB model must be efficient. It has minimum data to transfer between client and server
 - Pack only required data transferred between client and server
 - No redundant member object within an object

2.2 The HDF-SRB architecture

The HDF-SRB model consists of four basic components: the client (HDF client application or SRB client), the HDF-SRB module, SRB server, and the HDF library. Figure 3 illustrates the basic architecture of the HDF-SRB model.

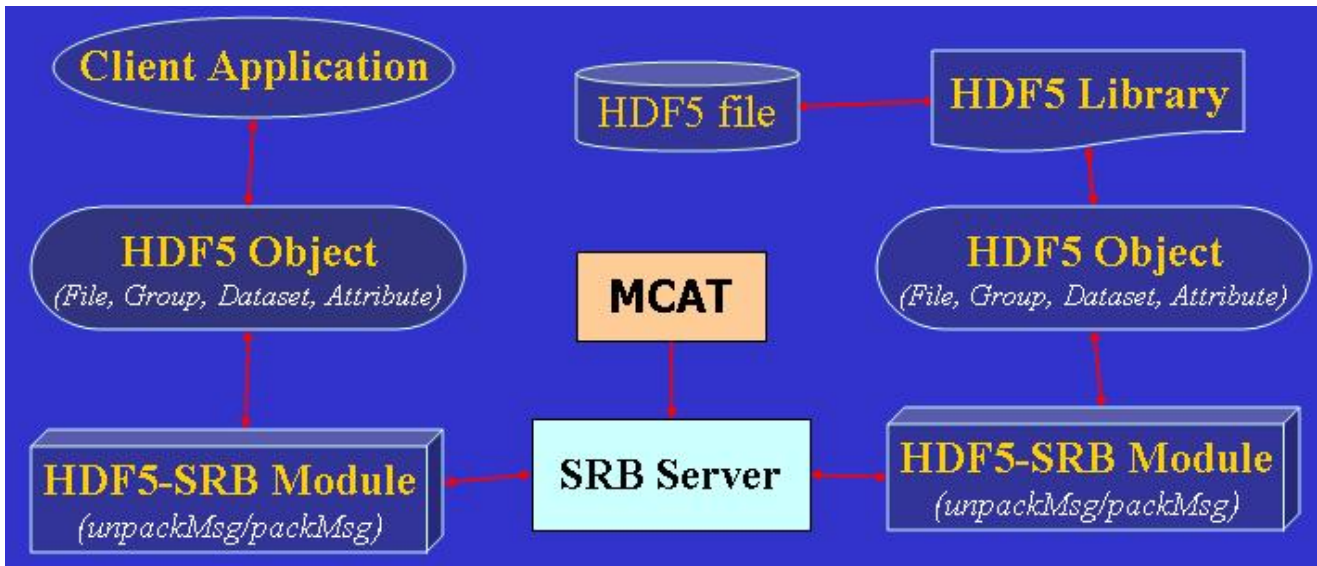


Figure 3 – HDF-SRB model

Client applications are implemented using a set of APIs provided by SRB for sending requests and receiving responses to/from the SRB servers. The requests and responses are packed with HDF objects. The critical component is the HDF-SRB module, which connects the HDF clients to the HDF library on the server. The HDF-SRB module is responsible for packing and unpacking messages, or HDF objects, between the SRB and HDF components. The HDF library is installed with the SRB server for interactive access to HDF files on the server side.

2.3 The data objects

In the HDF-SRB model, data objects are passed between the client and server instead of the entire files. There are several reasons for object level access. First, passing objects is more efficient than passing an entire file as passing a large file through the network can be very time consuming. It is sometimes not practical to transfer files with size of terabytes. For example, if we can just access the metadata in a terabyte-size file, then, select a small subset of the data, we can reduce the network transfer to a few megabytes. Second, it is easy to pass complex requests such as sub-setting. Because messages passing between the client and server are packed in data objects, there is no need to specify the format of the messages; messages, simple or complex, are self-explained in the object. Thirdly, it is easy to maintain and extend. Adding a new function to the object will not require any change to the data model

There are three basic data objects: H5File, H5Dataset and H5Group. H5File is used to hold metadata about the file and the root group of the file. The file structure can be by following the links that flow from the root group. H5Dataset contains data values and information about the data, such data type and data space. H5Attribute is similar to H5Dataset, but contains user-defined metadata for groups and datasets. H5Datatype contains information about the data type of the dataset, such as data type class, order and size. H5Dataspace contains sizes of the dimensions of dataset. It is also used for the client to passing information for sub-setting. The following sections will discuss the data objects in details.

Each data object has opID field. The opID is the operation identifier of the client request passed to the server. Based on the operation identifier, the server will take appropriate action on the file. For example, H5DATASET_READ operation calls the HDF5 library to read the data from the file.

H5File

H5File contains basic information of a file such as the full path of the file and the root group.

```
typedef struct H5File
{
    int      opID;          /* the id of operation to take on the server */
    char    *filename;     /* the name of the remote file */
    int      fid;          /* the file ID from H5Fopen() */
    H5Group *root;        /* the root group that holds the whole file structure */
    H5Error  error;       /* strings used to carry error report from server operation */
} H5File;
```

- ✓ **filename:** The filename is the SRB file name, a unique file name within a SRB storage. When you select a file from SRB storage, you get a SRB file with a logic SRB file path plus the user defined file name. For example, in /A/home/srbAdmin.demo/test.h5, /A/home/srbAdmin.demo is the logic file path in SRB and test.h5 is the user file. The logic file path does not have any meaning to the client, it is used for the server to map the SRB file to the local file that is the physical file on the server.
- ✓ **fid:** The fid is the file identifier from H5Fopen(). The file is open until H5Fclose(fid) is called. The fid is kept in all the object structure so that the file is open only once and file information is cached on the server for better performance.
- ✓ **root:** The root is the root group of the file. When client send “FILE_OPEN” request to the server, the server will call H5Fopen() and retrieve the file structure stored in the group structure and send it to the client. By traversing the group structure starting at the root group, the client will be able to construct the file structure.
- ✓ **opID:** The opID is the file operation identifier. The file operation identifier tells the server what operation the client have requested. The following is a list of valid operations on file:
 - FILE_OPEN: This operation will call H5Fopen() and retrieve the structure of the file into groups. By traversing the group structure starting at the root group, the client will be able to construct the file structure.
 - FILE_CLOSE: This will cause to server to call H5Fclose() to close all the resources on the server side.
 - FILE_CREATE: Create a new file on the server (not supported)
 - FILE_DELETE: Delete an existing file on the sever (not supported)

H5Group

H5Group holds the structure of one group in the file. The H5Group struct contains the full path of the group in the file, a list of group members (datasets and sub-groups), and an array of attributes.

```

typedef struct H5Group
{
    int          opID;          /* the id of operation to take on the server */
    int          fid;          /* the file ID from H5Fopen() */
    unsigned long objID[2];    /* uniquely object identifier [from server] */
    char         *fullpath;    /* the path + name, e.g. /hdfeos/swaths */
    struct H5Group *parent;    /* the parent group */
    int          ngroups;      /* number of group members in this group */
    struct H5Group *groups;    /* subgroup members of this group */
    int          ndatasets;    /* number of dataset members in this group */
    H5Dataset    *datasets;    /* dataset members of this group */
    int          nattributes;  /* number of attributes of this group */
    H5Attribute  *attributes;  /* attributes of this group */
    H5Error      error;        /* strings used to carry error report from server operation */
} H5Group;

```

- ✓ **fullpath:** The fullpath is the group full path within a file, the group path plus the group name. For example, /hdfeos/swaths
- ✓ **groups:** An array of group members.
- ✓ **datgaset:** An array of dataset members.
- ✓ **Attributes:** An array of attributes attached to the group.
- ✓ **opID:** Group operation identifier. The operation identifier tells the server what operation the client have requested. The following is a list of valid operations on groups:
 - GROUP_CREATE: Create a new group in the file (not supported).
 - GROUP_DELTE: Delete a group from the file (not supported).
 - GROUP_ATTR_READ: Retrieve all attributes attached to the group.
 - GROUP_ATTR_WRITE: Write an attribute value to the group (not supported).
 - GROUP_ATTR_CREATE: Attach a new attribute to the group (not supported).
 - GROUP_ATTR_DELETE: Delete an attribute from the group (not supported).

H5Dataset

The H5Dataset contains information of a dataset, such as the full path of the dataset, the datatype and dataspace, data that has been read from or is to be written to the dataset, and an array of attributes.

```

typedef struct H5Dataset
{
    int          opID;          /* the id of operation to take on the server */
    int          fid;          /* the file ID from H5Fopen() */
    unsigned long objID[2];    /* uniquely identify an object in a file [from server] */
    H5Datatype_class_t tclass; /* same as class in H5Datatype. Put here for packing */
    char        *fullpath;     /* the path + name, e.g. /hdfEOS/swaths/data fields/channel */
    int         nattributes;    /* number of attributes */
    H5Attribute *attributes;    /* array of attributes */
    H5Datatype  type;          /* the data type */
    H5Dspace    space;         /* the data space */
    unsigned int nvalue;       /* the number of bytes in the buffer *value */
    void        *value;       /* the data read from file or write to file */
    H5Error     error;        /* strings used to carry error report from server operation */
} H5Dataset;

```

- ✓ **fullpath:** The fullpath is the dataset full path within a file, e.g. /foo/datasets/d1
- ✓ **type:** The data type.
- ✓ **space:** The data space. The client will use the space information to determine the data size.
- ✓ **value:** The value is the values of the dataset retrieved from the file. The server calls H5Dread() and fill the value buffer with data from the file. Since the client cannot interpret complex data such as compound dataset and variable length dataset, the data values are converted into strings on the server before they are sent to the client.
- ✓ **Attributes:** An array of attributes attached to the dataset.
- ✓ **opID:** Dataset operation identifier. The operation identifier tells the server what operation the client have requested. The following is a list of valid operations on datasets:
 - DATASET_READ: Read an entire dataset or a subset.
 - DATASET_WRITE: Write data to the dataset (not supported).
 - DATASET_CREATE: Create a new dataset in the file (not supported).
 - DATASET_DELETE: Delete a dataset from the file (not supported).
 - DATASET_ATTR_READ: Retrieve all attributes attached to the dataset.
 - DATASET_ATTR_WRITE: Write an attribute value to the dataset (not supported).
 - DATASET_ATTR_CREATE: Attach a new attribute to the dataset (not supported).
 - DATASET_ATTR_DELETE: Delete an attribute from the dataset (not supported).

2.4 The client and server APIs

The SRB client/server software needs to be enhanced to support HDF5 because of the very complex nature of the HDF5 data structures and request characteristics. The enhancements include the pack/unpack routines, new proxy functions for server and client.

Pack/Unpack routine enhancements

The SRB pack/unpack routines must be enhanced to accommodate the HDF5 data structures. In SRB, the pack/unpack routines are used to send and receive data structure between clients and server. For example, for a typical client request, the input data structure is packed into a single byte stream and sends

to the server which in turns unpacks the byte stream it received back into the input data structure before the request is processed. The pack/unpack routines are also used to send and receive the output data structure from the server to client. Two upgrades to the pack/unpack routines were needed to handle the HDF5 data structure:

- ✓ Handles data structures containing deeply nested data structures. An example is the H5Group struct which may contain an array of H5Group struct as its sub-groups. Each one of these sub-groups may again contain their own sub-groups forming an hierarchical tree of group and sub-groups. The SRB pack/unpack routines are now able to drill down and recursively pack and unpack hierarchical structures.
- ✓ Handles data structures containing members with data types depending on the setting of other members. For example, the H5Dataset struct contains a generic member "void* value". "value" can be a pointer to an array of string or an array of character depending on the setting of another member - "class". If the value of "class" is set to H5DATATYPE_STRING (3), H5DATATYPE_COMPOUND (6) or H5DATATYPE_VLEN (9), "value" is a pointer to an array of string. Otherwise, "value" is a pointer to an array of character. The SRB pack/unpack routines are now able to pack and unpack structures with dependent data type based on some new syntax in the packing instruction.

General proxy function

We implemented a new and more general SRB proxy function framework to make it easier to implement and handle HDF5 requests which can be quite complex. On the client side, a user calls a new client API - `srbGenProxyFunc()` to make proxy request. One of the input parameter of this function - "funcType" specifies the type of proxy function being requested.

On the server side, the `svrGenProxyFunc()` function is used to dispatch the proxy function call. It uses the `genProxyFuncEntries[]` table, which is defined in `genProxyFunc.h` to determine which proxy function to call to handle the request. Currently, the `genProxyFuncEntries[]` is defined as follows:

```
genProxyFunc_t genProxyFuncEntries[] = {
    {HDF5_OPR_TYPE, (func_ptr) h5ObjProcess},
};
```

The table contains only one entry, the HDF5 type (HDF5_OPR_TYPE) proxy function. The `h5ObjProcess()` function will be called to handle the HDF5_OPR_TYPE request. To implement additional types of proxy functions, one needs to simply add more entries to the `genProxyFuncEntries[]` table and functions to handle these type of requests on the server.

The prototype for the client API - `srbGenProxyFunc()` is given as follows:

```
int srbGenProxyFunc (
    srbConn *conn, int funcType, int intInput1, int intInput2,
    char *strInput1, char *strInput2, void *inputStruct, FormatDef inputFormat,
    void **outputStruct, FormatDef outputFormat, FormatDefConst *localDef,
    int maxOutSz)
```

where

`srbConn* conn` - From `srbConnect()`.

int funcType -The type of proxy operation. e.g., HDF5_OPR_TYPE
int intInput1 - an arbitrary integer input that will be passed to the
proxy function handler on the server side.
int intInput2 - an arbitrary integer input.
char *strInput1 - an arbitrary string input.
char *strInput2 - an arbitrary string input.
void *inputStruct - Pointer to input struct.
FormatDef inputFormat - packing format for inputStruct.
void *outputStruct - Pointer to output struct.
FormatDef outputFormat - packing format for outputStruct;
FormatDefConst *localDef - Local Format definition. NULL means no Local
Local Format definition.
int maxOutSz - the max size of packed output

Packing of the input struct and unpacking of the output struct are done automatically in this routine. The "inputFormat" specifies the instruction for packing the input struct and the "outputFormat" specifies the instruction for unpacking the output struct.

The srbGenProxyFunc() client API and the server proxy function handling routine form a simple and flexible framework for implementing SRB proxy functions.

3 Client and Server Applications

In this chapter, we will discuss the HDF-SRB source code, instruction on how to setup a server and how to write a client application. As client application, we will give two examples, a standalone C application and an Java application implemented in the NCSA HDFView.

3.1 The source code

The HDF-SRB source code is part of the SRB source distribution version 3.4.x or above. The HDF-SRB source code is located at SRB3_4_x/proxy/hdf5. The source code is organized as following:

```

proxy
  hdf5
    include
      h5Attribute.h
      h5Dataset.h
      h5Dataspace.h
      h5Datatype.h
      h5File.h
      h5Group.h
      h5Handler.h
      h5Object.h
      h5String.h
      hdf5PackDefExtern.h
      hdf5PackDef.h
    common
      h5Ctor.c
      h5Dtor.c
    client
      clH5Dataset.c
      clH5File.c
      clH5Group.c
      h5ClHandler.c
      h5LocalHandler.c
    server
      h5Attribute.c
      h5Dataset.c
      h5File.c
      h5Group.c
      h5Object.c
      h5String.c
      h5SvHandler.c
    test
      test_h5File.c
      Makefile

```

The “include” directory contains all the header files needed by the client and the server. The “common” directory has two files: h5Ctor.c and h5Dtor.c. h5Ctor.c contains code for initializing the data structures

and h5Dtor is for clean up memory for the data structure. The “client” directory contains code only needed by the client such as the client handlers. The “server” directory is the server code, which include server handlers and implementation of the data structures. The test_h5File.c is a C example of how to write HDF-SRB client application. It is also used to test the HDF-SRB model.

3.2 Configuring and compiling the SRB server

This section gives instruction for configuring and compiling the SRB server code to support HDF5/SRB interface.

Currently, we don't have a way to configure HDF5 using the install.pl script for the combine installation of the Postgres MCAT and SRB software. However, the SRB software alone can be configured and built to support HDF5.

Configure and build HDFSRB from scratch

To configure and build the SRB client/server software that supports HDF5 from scratch, the following steps should be taken:

- a) Download the source from <http://www.sdsc.edu/srb/>, decrypt and untar the SRB software.
- b) Download and install pre-compiled zlib, szip and hdf5 library from <http://hdf.ncsa.uiuc.edu/HDF5/release/obtain5.html>.
- c) cd to the main SRB directory. e.g., "cd SRB3_4"
- d) Configure the SRB software to support HDF5 by typing in:

```
configure --enable-hdf5 \
    --enable-hdf5home=<hdf5path> \
    --enable-zlib=<zlibpath> \
    --enable-zip=<szippath>
```

where, hdf5path is the directory where the hdf5 library is installed. zlibpath and szip is where the zlib and szip installed respectively. For example,

```
configure --enable-hdf5 \
    --enable-hdf5home=/home/srb/hdf5-1.6.4/build\
    --enable-szip=/home/srb/lib_external/zlib121-linux2.4\
    --enable-zlib=/home/srb/lib_external/szip2.0-linux-enc
```

- e) Type in "gmake" to make the software, and follow the SRB instruction on how to start the server.

Configure and build HDFSRB from installed SRB

If the SRB software has been built using the install.pl script, the SRB software needs to be rebuilt in order to support HDF5. For example,

- a) cd to the main SRB directory. e.g., "cd SRB3_4"
- b) Type in “gmake clean”

To configure the SRB software to support HDF5, the configure command line options given in the config.log file plus the hdf5 option

```
--enable-hdf5 \
--enable-hdf5home=<hdf5path> \
```

- ```
--enable-zlib=<zlibpath> \
--enable-zip=<zippath>
```
- c) Type in "gmake" to make the software.

It should be noted that before starting the SRB server, the HDF5 library path should be included in the LD\_LIBRARY\_PATH path. e.g.,

```
setenv LD_LIBRARY_PATH /home/srb/hdf5-1.6.4/build/lib
```

### 3.3 Configuring, writing and compiling the client code

The test client program given in the proxy/hdf5/test directory can be used to illustrate the steps needed to compile and build a client program using the HDF5/SRB interface.

#### test/Makefile

The Makefile given in the proxy/hdf5/test directory should be used as a template to build a client program. The make requires that the client to have access to the SRB source tree and build environment. The following modifications may be required:

- a) a) The definition of "srbBuildDir" given at the beginning of the file "srbBuildDir" is the directory where the SRB server is built. Currently it is set to "../.." since the test program is built under the same build tree. This directory can be set to an absolute path.
- b) b) The steps that are normally taken when configuring the Makefile - replacing the names of .c, .o and executable files with your own files. e.g., replacing test\_h5File.c, test\_h5File.o and t5 with your file names.

#### Client header files

The client program should contain the following header files:

```
#include "h5File.h"
#include "h5Handler.h"
#include "hdf5PackDef.h"
```

Before the client program can be run, the SRB client user environment should be set up first. e.g., setting up the ~/.srb/.MdasEnv and ~/.srb/.MdasAuth files. The client program will look for the server options to make server connection.

#### Example of .MdasEnv

```
mdasCollectionName '/A/home/srbAdmin.demo'
mdasCollectionHome '/A/home/srbAdmin.demo'
mdasDomainName 'demo'
mdasDomainHome 'demo'
srbUser 'srbAdmin'
srbHost 'server.host.machine'
srbPort '5544'
defaultResource 'demoResc'
```

AUTH\_SCHEME 'ENCRYPT1'

Example of . MdasAuth

srbPasswd2005

### 3.4 HDFView – the Java client application

The HDFView is a visual tool for browsing and editing NCSA HDF4 and HDF5 files. Using HDFView, you can

- ✓ view a file hierarchy in a tree structure
- ✓ create new file, add or delete groups and datasets
- ✓ view and modify the content of a dataset
- ✓ add, delete and modify attributes
- ✓ replace I/O and GUI components such as table view, image view and metadata view

For more details on HDFView, visit the NCSA HDFView webpage at <http://hdf.ncsa.uiuc.edu/hdf-java-html/hdfview/index.html>.

Supporting HDF-SRB in HDFView requires implementing HDF-SRB Java Native Interface(JNI) and adding new GUI components and data object.

#### The HDF-SRB JNI

The Java language defines its own virtual machine and instruction set. This design makes Java easy to program and very portable. However, Java programs cannot use code written in any language except Java.

In order to provide a minimal level of interoperability with other languages, the Java Native Interface (JNI) defines a standard for invoking C and C++ subroutines as Java methods (declared 'native'). The Java Virtual Machine (JVM) implements a platform-specific mechanism to load C or C++ libraries, and to call the native methods on behalf of the Java programs.

So, to access a C library from a Java program, four tasks must be completed:

- a) Declare appropriate 'native' methods in one or more Java classes
- b) Write 'glue' code for each native method to implement the JNI C or C++ standard
- c) compile the JNI glue (C or C++) to create a JNI library
- d) deploy the Java classes and the JNI library on the system

When the Java program runs, the JNI glue and C library must be in the 'java.library.path' path.

The HDF-SRB JNI consists of an Java class and dynamically linked native library. The Java class declares static native methods, and the library contains C functions which implement the native methods. The C functions call the standard HDF-SRB client module.

The HDF-SRB JNI class contains only one native interface, h5ObjRequest(). h5ObjRequest () does two things: load the dynamic library and pass client requests to the C function.

```
public synchronized static native int h5ObjRequest
(String srb_info[], Object obj, int obj_type) throws Exception;
```

The dynamic library (C implementation of the native interface) wraps the SRB client and converts data object between C and Java. When client calls the Java interface `h5ObjRequest()`, the dynamic library does the following tasks:

- a) Make connection to the SRB server
- b) Decode the Java object and construct C structure
- c) Send requests to the server in the form of C structure
- d) Encode server result in to Java object

## The Java HDF-SRB objects

HDFView is implemented on the HDF Object Package. The HDF Object Package is a Java package which implement HDF4 and HDF5 data objects in an object-oriented form. The HDF Java Object Package provides a common standard Java API to access both HDF4 and HDF5 files.

The HDF Object Package, `ncsa.hdf.object`, provides classes that reflect fundamental conceptual to design of HDF objects. Objects of HDF5 (group and dataset) and HDF4 (group, multi-dimension array, raster image, vdata and annotation) are presented as Java classes. It has three major goals:

- a) simplifies the process of reading information from or writing data to file.
- b) hides the HDF4 and HDF5 libraries from applications.
- c) provides modular HDF4 and HDF5 packages. Applications can be built by using only the common object/interface layer

For more information on the HDF Object Package, visit <http://hdf.ncsa.uiuc.edu/hdf-java-html/hdf-object/index.html>.

To support HDF-SRB data objects, we have implemented the following Java package, `ncsa.hdf.srb.obj`, which is inherited from the common object layer.

- ✓ `H5SrbFile` *extends* `FileFormat`
- ✓ `H5SrbGroup` *extends* `Group`
- ✓ `H5SrbScalarDS` *extends* `ScalarDS`
- ✓ `H5SrbCompoundDS` *extends* `CompoundDS`
- ✓ `H5SrbDatatype` *extends* `Datatype`

These objects implement methods to deal with the client requests and data from the server. The native call, `h5ObjReques()`, pass the information through the objects. For example, the following code is how the we read data from remote file using `H5SrbScalarDS::read()`.

```

public Object read() throws Exception, OutOfMemoryError
{
 String srbInfo[] = ((H5SrbFile)getFileFormat()).getSrbInfo();
 if (srbInfo == null || srbInfo.length<5) return null;

 opID = H5DATASET_OP_READ;
 H5SRB.h5ObjRequest (srbInfo, this, H5SRB.H5OBJECT_DATASET);

 return data;
}

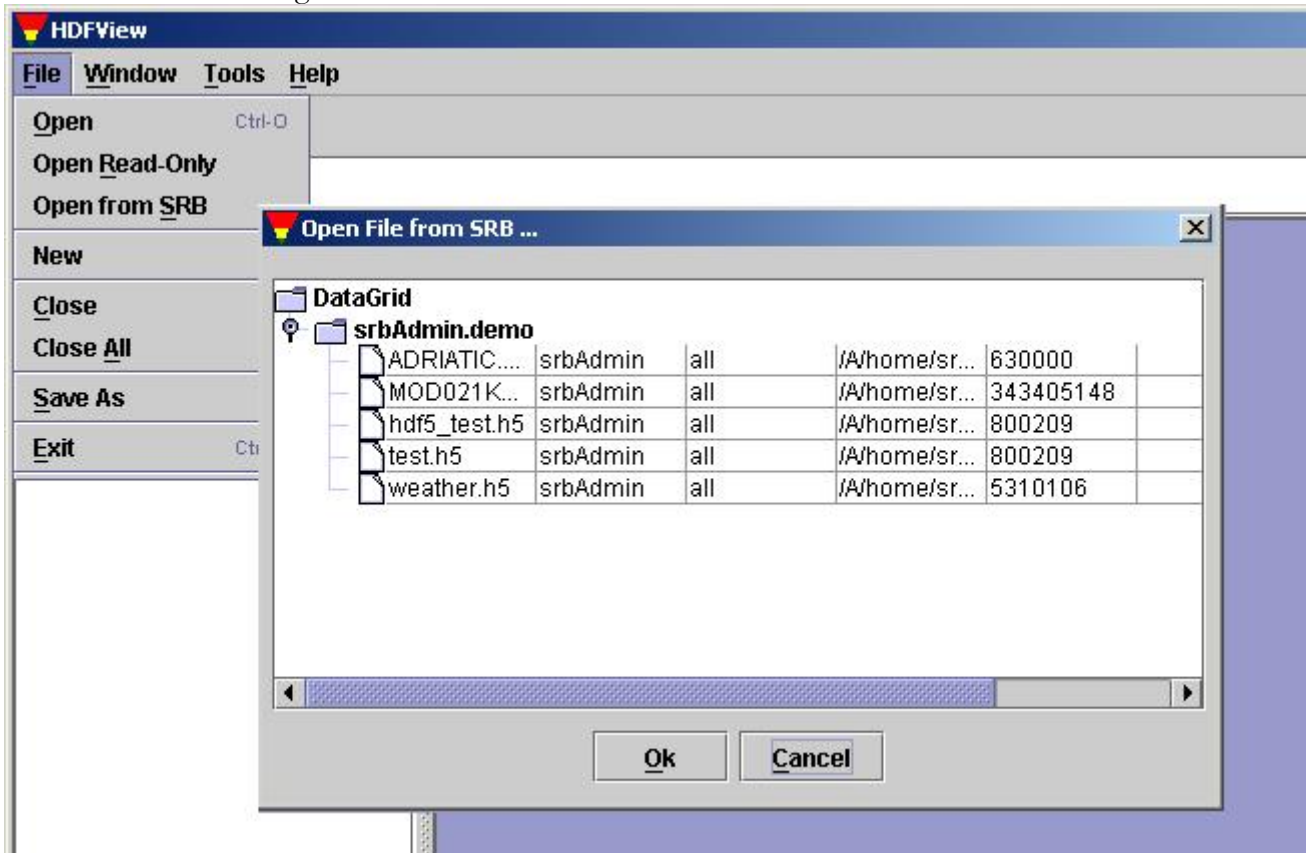
```

## The GUI components

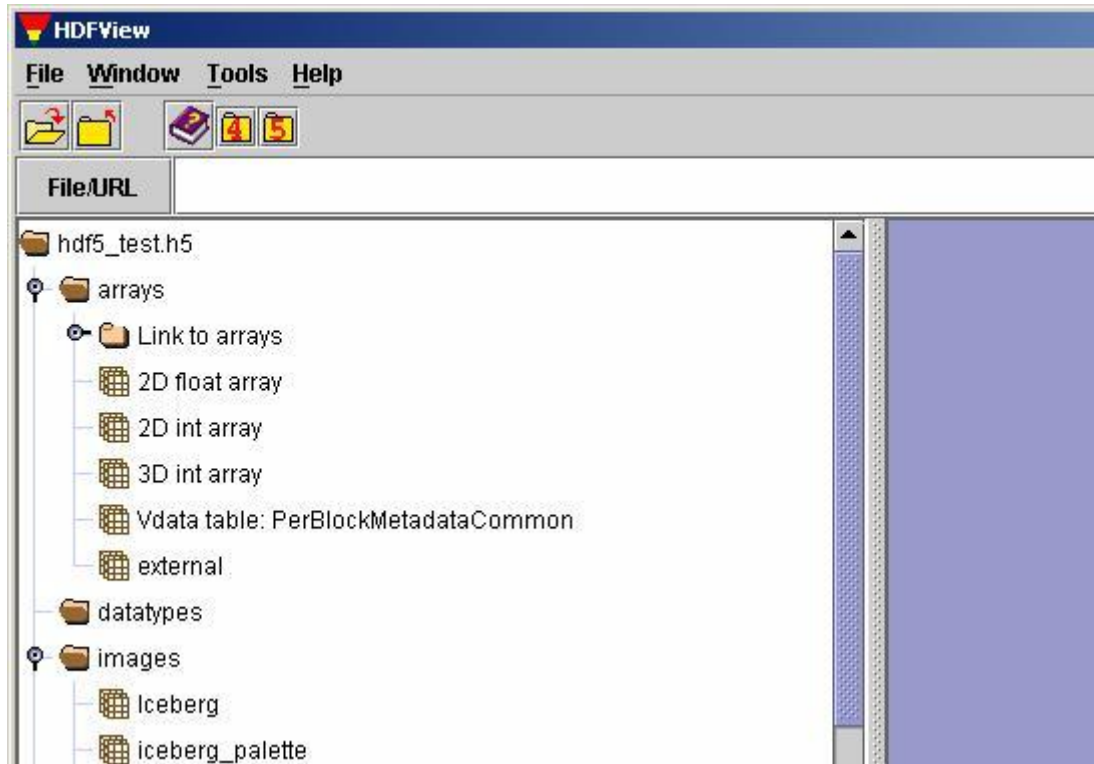
Since HDFView is built on modular fashion, the GUI components are transparent to data access. There is not much change to the GUI components. We added SRBFileDialog class to the GUI. SRBFileDialog is used to make connection to server through the Java Api for Real Grids On Networks (JARGON). JARGON is a pure java API for developing programs with a data grid interface. The API currently handles file I/O for local and SRB file systems, as well as querying and modify SRB metadata. For more information on JARGON, read <http://www.sdsc.edu/srb/jargon/index.html>.

You will go through the following steps to access file and data on SRB server.

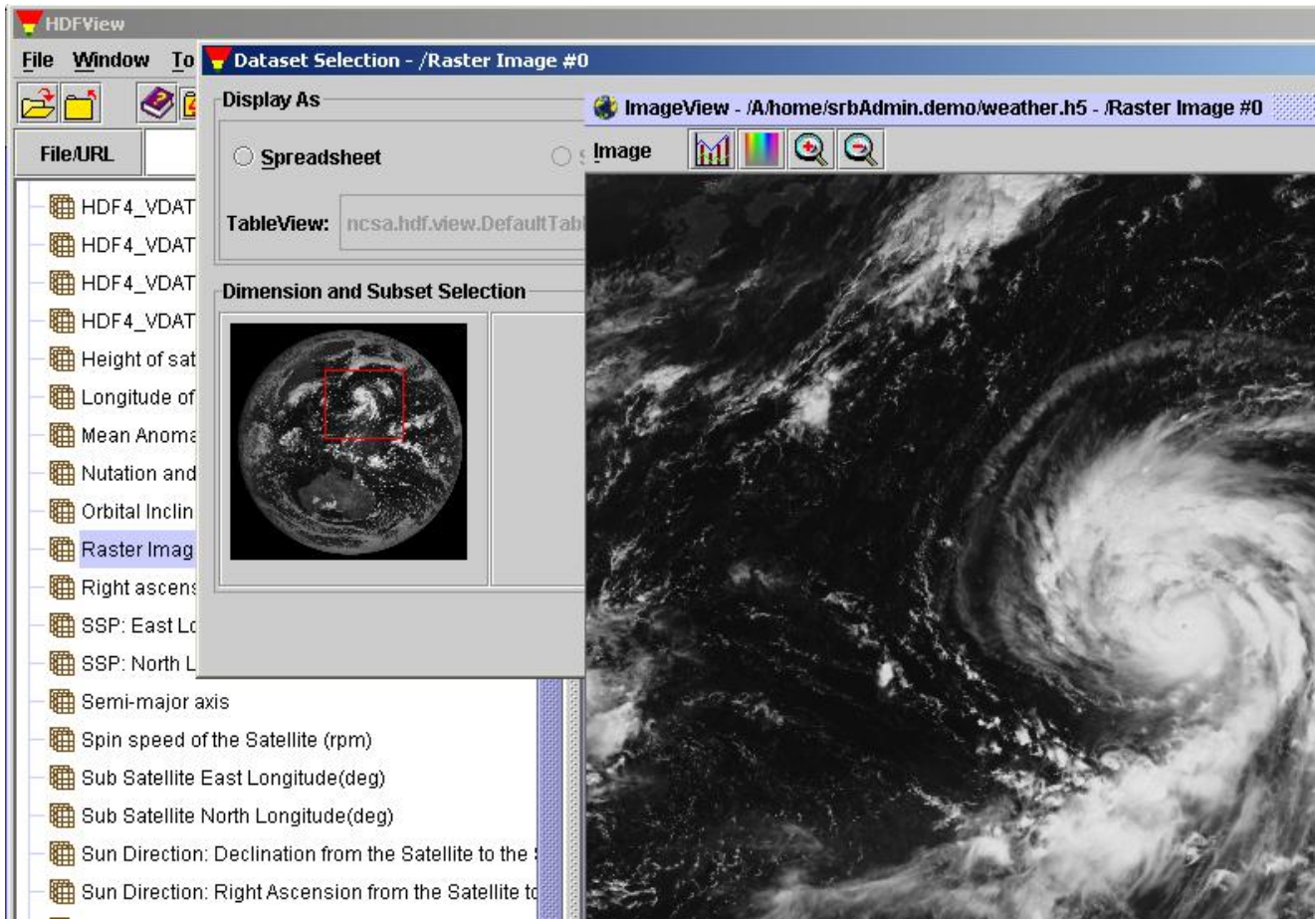
- a) Select “Open from SRB” from the “File” menu, which calls the Jargon APIs to make a server connection and gets a list of files on the server.



- b) Select a file from the file list to open. It creates an instance of `H5SrbFile` and calls the `H5SrbFile::open()`, which invokes the JNI call, `h5ObjRequest()`, to open the file at the server and request the file structure.



- c) Once you have the file structure in the HDFView, you can make other client requests on the file. For example, open a subset of a dataset.



Using HDFView, you can also open files from multiple SRB servers that support HDF5. To add server connections, you use the “User Options” from the “Tools” menu.

The image shows a 'User Options' dialog box with three tabs: 'General Setting', 'Default Module', and 'SRB Connection'. The 'SRB Connection' tab is active. On the left, there is a list of SRB connections with 'stanley.ncsa.uiuc.edu' selected. Below the list are 'Add' and 'Delete' buttons. On the right, there are several labeled text input fields: 'Host Machine' (stanley.ncsa.uiuc.edu), 'Port Number' (5544), 'User Name' (srbAdmin), 'Password' (srbdemo05), 'Home Directory' (/A/home/srbAdmin.demo), 'mdas Domain Name' (demo), and 'Default Storage Resource' (demoResc). At the bottom of the dialog are 'Ok' and 'Cancel' buttons.

**User Options**

**General Setting** **Default Module** **SRB Connection**

SRB Connections

- stanley.ncsa.uiuc.edu
- hdf2.incubator.uiuc.edu

**Add** **Delete**

**Host Machine:** stanley.ncsa.uiuc.edu

**Port Number:** 5544

**User Name:** srbAdmin

**Password:** srbdemo05

**Home Directory:** /A/home/srbAdmin.demo

**mdas Domain Name:** demo

**Default Storage Resource:** demoResc

**Ok** **Cancel**