IBP - Internet Backplane Protocol: Infrastructure for Distributed Storage (V 0.2)

Wael R. Elwasif, James S. Plank, Micah Beck

Department of Computer Science
University of Tennessee
Technical Report CS-99-430
February 1999

`[elwasif, plank, mbeck]@cs.utk.edu`
`http://www.cs.utk.edu/~plank/IBP`

Introduction:

In this document, we present a description of IBP v0.2, a client-server tool for remote storage management. We present the IBP client interface along with a detailed description of the semantics involved in every client call. As of the date of this document, IBP has been developed on SUN Solaris OS, with possible ports to other OS's in the future.

The IBP client

The current implementation of IBP supports only sunchronized client requests, all client IBP calls will block pending completion (or failure) on the server(s) size. It is envisioned that in the future this restriction could be relaxed to allow non-blocking IBP calls to be made. In what follows, we describe the calls that constitute the IBP client interface. We present the C-language prototype of every call along with a detailed description of the data structures, success behavior and error conditions involved in that call. In keeping with UNIX tradition, failure of an IBP client call is indicated by a return value of **-1** , or **NULL**, with a special variable (*IBP_errno)* set to the appropriate error code.

IBP_allocate()

include "ibp_client.h"

IBP_cap_set IBP_allocate(char *targetHost, ulong_t size, IBP_attributes attr)

IBP_allocate() allocates a remote storage area on the host ***targetHost***. The allocated area has a maximum possible size of ***size*** bytes, and storage attributes defined by ***attr***.*IBP_attributes* is typeded to a pointer to *struct ibp_attributes* defined in "**ibp_base.h"** and which has the following format

```
typedef struct ibp_attributes {
        time_t duration;
        int    reliability;
        int    type;
} *IBP_attributes;
```

Where

durattion specifies the time at which the allocated storage area will be automatically purged from the pool of storage areas managed by the server. Time is specified in seconds since the epoch (as returned by UNIX's **date**() command). A value of **0** indicates permanent status for the allocated storage area (it'll be only purged when no more clients have read access to it, otherwise it will be kept alive ccording to the reliability property)

reliability is a flag that determines how reliable the allocated storage area will be. The current version of IBP supports two level of reliability

- **reliability = IBP_STABLE** which guarantees the existence of the allocated storage area until it is removed due to lack of readers as explained above.
- **reliability = IBP_VOLATILE** which declares the allocated area to be volatile, in the sense that the corresponding IBP server can reclaim storage allocated to this area whenever site administration and/or IBP server policy mandates such move. Stable storage is never reclaimed by IBP server as long as at least one client has read access to that storage.

type is a flag that determines the type of storage allocated. The current version of IBP supports two types of storage

- **type = IBP_BYTEARRAY** which treats the allocated area as a flat byte array. This will have the following implications on future accesses to that storage area
  - Requests for read to the allocated area will be denied if there are not enough data to satisfy the read request at the time the request is recieved by the IBP server.
  - Requests to write (append) to the allocated area will be denied if it leads to the total size of the storage area exceeding the maximum allowable size specified in *size*.
  - A maximum of one write operation can be activily writing to the storage area at any given time, other write requests recieved by the server are queued pending completion of the running write process.
  - No limit is imposed on the number of simultaneous read accesses to the storage area. In addition, due to the use of append-only semantics for write operations, a write operation can

be simultaneously active with any number of read operations to the same storage area.

- **type = IBP_FIFO** which causes the allocated storage area to be treated as a FIFO queue, with the following implications:
  - Read data is removed from storage area once read .
  - Read requests will be blocked if not enough un-read data is available in the storage area. In addition, no upper limit is placed on the size of data in read requests.
  - Write requests will be blocked if there is not enough space in the storage area to complete the write operation. In addition, there is no upper limit on the size of data involved in a write operation to the storage area.
  - Blocked operations will be un-blocked only when there is more data to read (blocked read operation) or available space to write (blocked write operations)
  - A maximum of one write operation and one read operation can be simultaneously active at any given time. Further requests are blocked pending completion of running operations.
- **type = IBP_BUFFER** which causes the allocated storage area to be treated as a restricted-access flat storage area., with the following properties:
  - Only one process can be actively accessing the storage area for read and/or write operation at any given time. Other requests are blocked pending completion of the one that has access to the storage area at any given time.
  - All write operations start at the beginning of the storage area, overwriting any data that had been stored there previously (even if it had not been read).
  - The amount of data available to a read operation at any given time is the amount that had been stored by the last write call.

Return values

Upon success, *IBP_allocate()* returns an *IBP_cap_set* object ,wich is a pointer to *struct ibp_cap_set* defined in **"ibp_base.h",** and has the following format.

```
typedef char* IBP_cap;
typedef struct ibp_cap_set{
      IBP_cap readCap;
      IBP_cap writeCap;
      IBP_cap manageCap;
} *IBP_cap_set;
```

In the current version of IBP, IBP capability type (IBP_cap) is typedefined to be a simple character string. This however could change in future versions of IBP. The capabilities included in an *IBP_cap_set* object allow the client read access, write access, and management access to the newly created storage area, respectively.

Upon failure, **IBP_allocate()** returns a *NULL* pointer and sets *IBP_errno* to one of the following values defined in **"ibp_protocol.h"**

- **IBP_INVALID_PARAMETER**: One or more of the parameters to the IBP_allocate() call has an invalid value (e.g. NULL *targetHost,* invalid entry in *attr,* ..etc.)
- **IBP_CONNECTION_ERROR**: An error has occured while trying to connect to the IBP server running on ***targetHost***.
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to the socket connection to the IBP server.

- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from the IBP server .
- **IBP_BAD_FORMAT**: Response from the IBP server does not have the expected format, or the IBP server recieved a badly formatted request.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_WOULD_EXCEED_LIMIT**: Granting the request would cause the IBP server to exceed the maximum storage limit allocated to the storage category defined in *attr*.
- **IBP_FILE_ACCESS_ERROR**: The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.
- **IBP_INTERNAL_ERROR**: The IBP server has encountered an internal error while processing the client's request.

IBP_store()

include "ibp_client.h"

int IBP_store(IBP_cap cap, char *data, ulong_t size)

IBP_store() stores ***size*** bytes starting at ***data*** at the storage area accessed through the IBP capability ***cap***. For this call to succeed, ***cap*** must be a *writecap* returned by an earlier call to **IBP_allocate(),** or imported from the client which made the **IBP_allocate()** call. **IBP_store()** is a blocking call that only returns when the required size of data is successfully stored at the desired storage area accessed through the IBP capability ***cap***, or an error causes the call to abort prematurely. The call appends data to the end of any previously stored data at the storage area accessed through ***cap*** for storage areas of type **IBP_BYTEARRAY** and **IBP_FIFO.** Data written to a storage area of type **IBP_BUFFER** overwrites any previous data (starting at the beginning of the buffer).

Return values

Upon success, **IBP_store()** returns **0**. Otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_WRONG_CAP_FORMAT**: The IBP capability ***cap*** doesn't have the proper format.
- **IBP_CAP_NOT_WRITE**: The IBP capability ***cap*** is not a write capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to connect to the IBP server running on ***targetHost.***
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to the socket connection to the IBP server.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from the IBP server, or on the server side while reading transferred data from the client.
- **IBP_BAD_FORMAT**: Response from the IBP server does not have the expected format, or the IBP server recieved a badly formatted request.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: The storage area accessed through ***cap*** does not exist on the associated IBP server.
- **IBP_CAP_ACCESS_DENIED**: The storage area accessed through ***cap*** cannot be accessed for

write operations.

- **IBP_SIZE_EXCEEDS_LIMIT**: The write operation would cause the aggregate size of the storage area to exceed the maximum size specified in the **IBP_allocate()** call. This error is only relevant for storage areas of type *IBP_BYTEARRAY*.
- **IBP_FILE_ACCESS_ERROR**: The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.
- **IBP_FILE_WRITE_ERROR**: The IBP server has encountered an error while attempting to store incoming data to the underlying storage area.
- **IBP_RESOURCE_BUSY**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.
- **IBP_INTERNAL_ERROR**: The IBP server has encountered an internal error while processing the client's request.

IBP_remote_store()

#include "ibp_client.h"

int IBP_remote_store(IBP_cap cap, char *host, ushort_t port, ulong_t size)

IBP_remote_store() causes the IBP server that hosts the storage area accessed through the IBP capability ***cap*** to fetch ***size*** bytes from a socket connection to ***port*** on ***host.*** Data fetched is then written to the storage area accesssed through *cap,* in a manner similar to that described in the **IBP_store()** call. It is the responsibility of the client(s) to make arrangements for the specified amount of data to be served at the given port. This call is a blocking call that returns only when data transfer from the remote host to the IBP server is completed successfully, or terminated due to an error condition. For this call to succeed, ***cap*** must be a *writecap* returned by an earlier call to **IBP_allocate(),** or imported from the client which made the **IBP_allocate()** call.

Return values

Upon success, **IBP_remote_store()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_WRONG_CAP_FORMAT**: The IBP capability ***cap*** doesn't have the proper format.
- **IBP_CAP_NOT_WRITE**: The IBP capability ***cap*** is not a write capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to connect to the IBP server hosting the storage area accessed through ***cap,***or an error ocured when the IBP server attempted to establish a connection with the remote data host.
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to the socket connection to the IBP server.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from the IBP server, or on the server side while reading transferred data from the remote data host.
- **IBP_BAD_FORMAT**: Response from the IBP server does not have the expected format, or the IBP server recieved a badly formatted request.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: The storage area accessed through ***cap*** does not exist on the

associated IBP server.

- **IBP_CAP_ACCESS_DENIED**: The storage area accessed through ***cap*** cannot be accessed for write operations.
- **IBP_SIZE_EXCEEDS_LIMIT**: The write operation would cause the aggregate size of the storage area to exceed the maximum size specified in the **IBP_allocate()** call. This error is only relevant for storage areas of type *IBP_BYTEARRAY*.
- **IBP_FILE_ACCESS_ERROR**: The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.
- **IBP_FILE_WRITE_ERROR**: The IBP server has encountered an error while attempting to store incoming data to the underlying storage area.
- **IBP_RESOURCE_BUSY**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.
- **IBP_INTERNAL_ERROR**: The IBP server has encountered an internal error while processing the client's request.

IBP_read()

#include "ibp_client.h"

int IBP_read(IBP_cap cap, char *buf, long size, ulong_t offset)

IBP_read() reads ***size*** bytes, starting at ***offset***, from the storage area accessed through the IBP capability **cap**, into memory pointed to by **buf**. For storage areas of type *IBP_FIFO*, ***offset*** is ignored. A ***size*** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area to be read. For storage areas of type *IBP_FIFO*, a ***size*** value of **-1** causes a read operation for the maximum size specified in **IBP_allocate()** to be initiated. For this call to succeed, ***cap*** must be a *readcap* returned by an earlier call to **IBP_allocate(),** or imported from the client which made the **IBP_allocate()** call. **IBP_read()** is a blocking call that returns only when all required data is read, or the read operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_read()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_INVALID_PARAMETER**: One or more of the parameters to the **IBP_ead()** call has an invalid value (e.g. negative ***size***, ..etc.)
- **IBP_WRONG_CAP_FORMAT**: The IBP capability ***cap*** doesn't have the proper format.
- **IBP_CAP_NOT_READ**: The IBP capability ***cap*** is not a read capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to connect to the IBP server hosting the storage area accessed through ***cap.***
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to the socket connection to the IBP server involved.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from the IBP server.

- **IBP_BAD_FORMAT**: Response from the IBP server does not have the expected format, or the IBP server recieved a badly formatted request.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: The storage area accessed through ***cap*** does not exist on the associated IBP server.
- **IBP_CAP_ACCESS_DENIED**: The storage area accessed through ***cap*** cannot be accessed for write operations.
- **IBP_INSUF_DATA_ERROR**: There does not exist enough stored data to satisfy the read part of the copy request. This error is only relevant for storage areas of type *IBP_BYTEARRAY*.
- **IBP_FILE_ACCESS_ERROR**: One of the IBP servers has encountered an error while trying to access one or more of its internal files that control access to the storage area.
- **IBP_FILE_READ_ERROR**: The target IBP server has encountered an error while attempting to read data from underlying storage area.
- **IBP_RESOURCE_BUSY**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.
- **IBP_INTERNAL_ERROR**: The IBP server has encountered an internal error while processing the client's request.

IBP_copy()

#include "ibp_client.h"

int IBP_copy(IBP_cap source, IBP_cap target, long size, ulong_t offset)

IBP_copy() copies ***size*** bytes, starting at ***offset***, from the storage area accessed through the IBP read capability ***source*** and writes them to the storage area accessed through the IBP write capability ***target***. For storage areas of type *IBP_FIFO*, ***offset*** is ignored. A ***size*** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area accessed through ***source*** to be copied. For source storage areas of type *IBP_FIFO*, a ***size*** value of **-1** causes a copy operation for the maximum size specified in **IBP_allocate()** to be initiated. As in other read operations to an IBP_FIFO type storage area, data read from the storage area will no longer be available for future reads. For this call to succeed, **source** must be a *readcap* returned by an earlier call to **IBP_allocate(),** or imported from the client which made the **IBP_allocate()** call and ***target*** must be a *writecap* returned by a similar call.

IBP_copy() is a blocking call that returns only when all required data is successfully copied from the source IBP server to the target IBP server, or the operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_copy()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_INVALID_PARAMETER**: One or more of the parameters to the **IBP_copy()** call has an invalid value (e.g. negative ***size***, ..etc.)
- **IBP_WRONG_CAP_FORMAT**: One or both of the two capabilities does not have the proper

format.

- **IBP_CAP_NOT_READ**: The IBP capability ***source*** is not a read capability.
- **IBP_CAP_NOT_WRITE**: The IBP capability ***target*** is not a write capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to establish a connection to one (or both) of the IBP servers.
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to the socket connection to one or more IBP server.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from one or more IBP server.
- **IBP_BAD_FORMAT**: A badly formatted message has been recieved by an IBP server, or by the calling client.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: One (or both) of the storage areas involved in the copy operation does not exist on the associated IBP server.
- **IBP_CAP_ACCESS_DENIED**: One (or both) of the storage areas cannot be accessed for the required operation (read or write).
- **IBP_INSUF_DATA_ERROR**: There does not exist enough stored data to satisfy the read part of the copy request. This error is only relevant for source storage areas of type *IBP_BYTEARRAY*.
- **IBP_SIZE_EXCEEDS_LIMIT**: The write part of the copy operation would cause the aggregate size of the target storage area to exceed the maximum size specified in the **IBP_allocate()** call. This error is only relevant for target storage areas of type *IBP_BYTEARRAY*.
- **IBP_FILE_ACCESS_ERROR**: An IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area specified.
- **IBP_FILE_READ_ERROR**: The source IBP server has encountered an error while attempting to read data from underlying storage area.
- **IBP_FILE_WRITE_ERROR**: The targetIBP server has encountered an error while attempting to store incoming data to the underlying storage area.
- **IBP_RESOURCE_BUSY**: A resource used by one of the two IBP servers was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.
- **IBP_INTERNAL_ERROR**: An IBP server has encountered an internal error while processing the client's request.

IBP_deliver()

#include "ibp_client.h"

int IBP_deliver(IBP_cap source, char *targetHost, ushort_t port, long size, ulong_t offset)

IBP_deliver() delivers ***size*** bytes from the storage area accessed through the IBP capability ***source*** at offest ***offset*** to a waiting process running on host ***targetHost***and listening on port ***port***. For source storage areas of type *IBP_FIFO*, the parameter ***offset*** is ignored. A ***size*** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area accessed through ***source*** to be delivered. For source storage areas of type *IBP_FIFO*, a ***size*** value of **-1** causes a deliver operation for the maximum size specified in **IBP_allocate()** to be initiated. As in other read operations to an IBP_FIFO type storage area, data read from the storage area will no longer be available for future reads. For this call to succeed, **source** must be a *readcap* returned by an earlier call to **IBP_allocate(),** or imported from the client which made the **IBP_allocate()** call. It is the responsibility of the calling process to ensure the existence of a recepient process on ***targetHost*** that is listening on port ***port*** (this process will be serving a socket to which the IBP server will connect to initiate the delivery operation.) **IBP_deliver()** is a blocking call that only returns when the required amount of data is delivered to its destination, or the process is aborted due to an error.

Return values

Upon success, **IBP_deliver()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_INVALID_PARAMETER**: One or more of the parameters to the **IBP_deliver()** call has an invalid value (e.g. negative ***size***, ..etc.)
- **IBP_WRONG_CAP_FORMAT**: One or both of the two capabilities does not have the proper format.
- **IBP_CAP_NOT_READ**: The IBP capability ***source*** is not a read capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to establish a connection to the IBP server, or from the IBP server to the target host.
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to a socket connection that is part of the transaction.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from a socket connection that is part of the transaction.
- **IBP_BAD_FORMAT**: A badly formatted message has been recieved by an IBP server, or by the calling client.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: The source storage areas does not exist on the associated IBP server.
- **IBP_CAP_ACCESS_DENIED**: The source storage areas cannot be accessed for read operation.
- **IBP_INSUF_DATA_ERROR**: There does not exist enough stored data in the source storage area to satisfy the deliver request. This error is only relevant for source storage areas of type *IBP_BYTEARRAY*.
- **IBP_FILE_ACCESS_ERROR**: The IBP server has encountered an error while trying to access one or more of its internal files that control access to the source storage area.
- **IBP_FILE_READ_ERROR**: The IBP server has encountered an error while attempting to read data from underlying source storage area.
- **IBP_RESOURCE_BUSY**: A resource used by the source IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.
- **IBP_INTERNAL_ERROR**: An IBP server has encountered an internal error while processing the client's request.

IBP_manage()

#include "ibp_client.h"

int IBP_manage(IBP_cap cap, int cmd, int capType, IBP_status info)

IBP_manage() allows an IBP client to perform certain management operations on an IBP storage area. Any client that can present the management capability can issue any of the management commands described below. ***cap*** is an IBP management capability that is returned in the **IBP_allocate()** call or imported from the client which made that call. ***cmd*** can take one of the following values (defined in the file **"ibp_protocol.h"**)

- **cmd = IBP_INCR** increments the reference count to the capability associated with the management capability ***cap***, and whose type is specified in the parameter ***capType.*** Parameter ***info*** is ignored for this command.
- **cmd = IBP_DECR** decrements the reference count to the capability associated with the management capability ***cap*** and whose type is specified in the parameter ***capType.*** Decrementing the reference count the read capability associated with a storage area to **0** causes the IBP server to jettison that storage area from its managed pool. Further requests to that area will fail. while requests currently in progress will be allowed to progress to completion. Parameter ***info*** is ignored for this command.
- **cmd = IBP_CHNG** changes one or more of the attributes of the storage area accessed through the management capability ***cap.***The new values are specified through the parameter ***info*** (described below) The current version of IBP allows changes to one (or more) of the following attributes:
  - ***maxSize*** changes the maximum storage size of the underlying storge area. Changing the size of a storage area of type *IBP_FIFO* is currently not allowed. Decreasing maximum size of a storage area of type *IBP_BYTEARRAY* does not affect data already stored there, it will only affect future requests to that storage area.
  - ***duration*** changes the duration property of the storage area (see description of the **IBP_allocate()** call for further details on the possible values and implications for this parameter.)
- **cmd = IBP_PROBE** checks the current state of the storage area accessed through the management capability ***cap***. The current state is returned through the parameter ***info***, which is defined below.

capType determines the type of the capability affected by the two commands *IBP_INCR* and *IBP_DECR*. It can have one of two values, *IBP_READCAP* and *IBP_WRITECAP*. It is ignored for the two commands *IBP_CHNG* and *IBP_PROBE*.

info is a pointer to a truct of type `struct ibp_status` (typedefed to IBP_status) The struct has the following format

```
typedef struct ibp_status{
   int      readRefCount;
   int      writeRefCount;
   int      currentSize;
   ulong_t maxSize;
   struct ibp_attributes attrib;
} *IBP_status;
```

where *readRefCount* and *writeRefCount* hold the reference count for the read and write capabilities respectively (on return from an *IBP_PROBE* command) and are ignored for other commands. *currentSize* holds the current size of data stored in the underlying storage area (for storage areas of type

IBP_FIFO, it holds the maximum size of the underlying storage area). *maxSize* holds the maximum size of the storage area, while *attrib* holds the storage area attributes as defined earlier.

The following table summarizes the use of different parameters with every command.

	capType	readRefCount	writeRefCount	currentSize	maxSize	attrib
IBP_INCR	In	Not used	Not used	Not used	Not used	Not used
IBP_DECR	In	Not used	Not used	Not used	Not used	Not used
IBP_PROBE	Not used	Out	Out	Out	Out	Out
IBP_CHNG	Not used	Not used	Not used	Not used	In	In

Return values

Upon success, **IBP_manage()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes

- **IBP_INVALID_PARAMETER**: One or more of the parameters to the **IBP_manage()** call has an invalid value (e.g. invalid ***capType***, *NULL**info*** with *IBP_CHNG* or I*BP_PROBE* commands, unrecognizable ***cmd***,..etc.)
- **IBP_WRONG_CAP_FORMAT**: The capability does not have the proper format.
- **IBP_CAP_NOT_MANAGE**: The capability is not a manage capability.
- **IBP_CONNECTION_ERROR**: An error has occured while trying to establish a connection to the IBP server.
- **IBP_SOCK_WRITE_ERROR**: An error has occured while trying to write to a socket connection that is part of the transaction.
- **IBP_SOCK_READ_ERROR**: An error has occured while trying to read response from a socket connection that is part of the transaction.
- **IBP_BAD_FORMAT**: A badly formatted message has been recieved by an IBP server, or by the calling client.
- **IBP_INVALID_CMD**: The IBP server has recieved a command it does not recognize.
- **IBP_CAP_NOT_FOUND**: The source storage areas does not exist on the associated IBP server.
- **IBP_INVALID_MANAGE_CAP**: The management cap does not match the management cap associated with the storage area.
- **IBP_WOULD_DAMAGE_DATA:** Trying to change the size of a storage area of type *IBP_FIFO.*
- **IBP_WOULD_EXCEED_LIMIT**: Trying to increase the maximum size of an *IBP_BYTEARRAY* type storage area leads to exceding the maximum storage space allocated for its class of storage.

The IBP server:

The IBP server manages access to a pool of storage areas that are created, accessed, and managed remotely through the IBP client interface. The IBP server performs no security checks on clients' requests. A client which connects to the IBP server with the proper capability is granted access to the underlying functionality. The only "protected" operations are those performed through the **IBP_manage()** call. This call requires the client to present the management capability that is returned as part of the **IBP_allocate()** call before the server can fulfill the client's request.

The current version of IBP supports two levels of reliable storage, with the client choosing the level of reliability of an allocated storage area.:

- **Stable storage**: Allocating a storage area with this reliability level guarantees the permanent presence of the storage area until the read reference counter to it drops to zero, at which time the data therein is removed and the allocated storage is reclaimed by the IBP server.
- **Volatile storage**: This is storage that could be reclaimed by the IBP server at any time. The decision to reclaim a volatile storage area (or part thereof) could be dectated by site management policies or changes in storage requirements of local jobs.

In addition to the aforementioned reliability levels, IBP supports indefinite storage, where storage areas are only reclaimed through client requests or reliability-induced server actions, and time-limited storage, in which the storage area is reclaimed by the IBP server at a certain instance of time specified by a managing IBP client (a managing IBP client is a client that allocates a storage area, or acquires the management capability from the creating client.) See description of the client **IBP_allocate()** and **IBP_manage()** calls for ways of controlling properties of storage areas.

IBP server configuration

The IBP server is configured through the configuration file **"ibp.cfg",** which should be located in the home directory of the user launching the IBP server. This file contains a list of the form

Configuration parameter name Parameter value

with one entry per line. The following table lists the currently supported configuration parameters, their names, types, and default values.

Parameter name (case sensitive)	Description	Type	Default value
VOLSIZE	Size of available storage for volatile storage areas (in MegaBytes).	Integer	0
VOLDIR	Directory where volatile storage areas are to be stored (absolute path)	String	/tmp/
STABLESIZE	Size of available storage for stable storage areas (in MegaBytes).	Integer	50
STABLEDIR	Directory where stable storage areas are to be stored (absolute path)	String	/tmp/
HOSTNAME	FQDN to be used by clients to connect to the IBP server	String	Use DNS to try retrieving this entry

Default values are used if the corresponding parameter is not specified in the **"ibp.cfg"** configuration file (or if the file does not exist in the home directory of the user launching the IBP server).

IBP server blocking rules

The IBP server processes requests on a First Come First Serve basis. Due to the fact that all write operations to storage areas have append semantics, a maximum of one write operation can be actively accessing a storage area at any given time. There is no limit on the number of read processes (for storage areas of type I*BP_BYTEARRAY*), and an upper limit of one read process for dtorage areas of type *IBP_FIFO*. If a write request is recieved while another one is active to the same storage area, the new request is queued pending completion of the existing request (the same aplies to multiple read requests to storage areas of type *IBP_FIFO*).