Internet Backplane Protocol: C API 1.4

Yong Zheng Alessandro Bassi Micah Beck James S. Plank Rich Wolski

Department of Computer Science
University of Tennessee
Knoxville, TN 37996

[yong,abassi,mbeck,plank,rich]@cs.utk.edu

Abstract

In this document, we present a description of the IBP version 1.4 C API. The current implementation of IBP supports only synchronized client requests; all client IBP calls will block pending completion on the server(s) side, or the expiration of the client's timeout. Failure of an IBP client call is indicated by a return value of **0**, or **NULL**, with a special variable (**IBP_errno**) set to the appropriate error code.

1 IBP Data Structures

A few data structures are used through the IBP world.

1.1IBP capability

This is the basic building block of IBP. Its format is:

`ibp://`*`hostname`*`:`*`port`*`/`*`resourceid#key`*`/`*`WRMKey`*`/`*`WRM`*

- **hostname** The host name of IBP depot.
- **port** The port number on which IBP depot is running.
- **resourceid** The resource identification on IBP depot
- **key** The key can be roughly considered to be the filename.
- **WRMKey** The WriteKey/ ReadKey/ ManageKey is a value that allows the access to the right key for writing, reading,and managing.
- **WRM** This field can have only three values, and is linked to the field above. The values are:
  - WRITE
  - READ
  - MANAGE

1.2 Various data structures

1.2.1 struct ibp_depot

```
typedef  struct ibp_depot {
#define IBP_MAX_HOSTNAME_LEN    256
            char        host[IBP_MAX_HOSTNAME_LEN];
            int         port;
            int         rid;
} *IBP_depot;
```

- **host :** host name of IBP depot.
- **port :** the port number on which IBP depot is running.
- **rid :** resource identification. From version 1.4, IBP depot supports multiple resources, each resource is assigned a unique integer called resource identification. A IBP depot must have a resource with resource id 0 called default resource.

1.2.2 struct ibp_attributes

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

- **duration :** 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 time(2) function). A value of -1 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 according to the reliability property, see **IBP_manage** () call for detail).
- **reliability :** is a flag that determines how reliable the allocated storage area will be. The current version of IBP supports two levels of reliability:

  – ***IBP_HARD*** which guarantees the existence of the allocated storage area until it is removed due to lack of readers as explained above.

  – ***IBP_SOFT*** 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 four types of storage:

  – *IBP_BYTEARRAY* which treats the allocated area as a flat bytearray. This will have the following implications on future accesses to that storage area:

  * Only the amount of available data will be returned if there are not enough data to satisfy the read request at the time the request is received 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 actively writing to the storage area at any given time; other write requests received 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.

- *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.
- *IBP_CIRQ* which causes the allocated storage area to be treated as a Circular 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 NOT be blocked if there is not enough space in the storage area to complete the write operation, but will overwrite the beginning of the queue. In addition, there is no upper limit 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).
  - * 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.
- *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 area doperation at any given time is the amount that had been stored by the last write call.

1.2.3 struct ibp_set_of_caps

```
#typedef char * IBP_cap ;
typedef struct ibp_set_of_caps {
            IBP_cap             readCap;
            IBP_cap             writeCap;
            IBP_cap             manageCap;
} * IBP_set_of_caps;
```

The capabilities included in an *IBP_set_of_caps* object allow the client read access, write access, and management access to a particular storage area, respectively.

1.2.4 struct ibp_capstatus

```
typedef struct ibp_capstatus {
            int                         readRefCount;
            int                         writeRefCount;
            int                         currentSize;
            int                         maxSize;
            struct ibp_attributes       attrib;
```

} * IBP_CapStatus;

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 the other *IBP_manage*() commands. **currentSize** holds the current size of data stored in the underlying storage area(for storage areas of type *IBP_FIFO* and *IBP_CIRQ* 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.

1.2.4 struct ibp_dptinfo

```
typedef struct ibp_dptinfo {
            unsigned long int           StableStor;
            unsigned long int           StableStorUsed;
            unsigned long int           VolStor;
            unsigned long int           VolStorUsed;
            long                        Duration;
            float                       majorVersion;
            float                       minorVersion;
            int                         rid;
            int                         type;
            long long                   HardConfigured;
            long long                   HardServed;
            long long                   HardAllocable;
            long long                   TotalConfigured;
            long long                   TotalServed;
            long long                   TotalUsed;
            long long                   SoftAllocable;
            int                         nDm;
            int                         *dmTypes;
            int                         nNFU;
            int                         *NFU;
} *IBP_DptInfo;
```

• **StableStor**	hard storage size in MBytes. A deprecated field.
• **StableStorUsed**	hard storage used in MBytes. A deprecated field.
• **VolStor**	total storage size in MBytes. A deprecated field.
• **VolStorUsed**	total used storage in MBytes. A deprecated field.
• **Duration**	maximum allowed duration on IBP depot.
• **majorVersion**	IBP depot major version number.
• **minorVersion**	IBP depot minor version number.
• **rid**	resource identification.
• **type**	resource type. Currently, IBP depot supports two types of resource :

– MEMORY	ram resource.
– DISK	disk resource.

• **HardConfigured**	total hard storage size specified by IBP depot administrator.
• **HardServed**	actual total hard storage size provided by IBP depot.
• **HardUsed**	hard storage used.
• **HardAllocable**	free hard storage size.
• **TotalConfigured**	total storage size specified by IBP depot administrator.
• **TotalServed**	actual total storage size provided by IBP depot.
• **TotalUsed**	total storage used.
• **SoftAllocable**	free soft storage size.
• **nDM**	number of data mover types supported by IBP depot.
• **dmTypes**	array of data mover types.
• **nNFU**	number of NFU operations supported by IBP depot.
• **NFU**	array of nfu operations.

From version 1.4, IBP depot reports more information than previous version does. **StableStor , StableStorUsed, VolStor and VolStorUsed** are deprecated fields and only used when IBP client communicates with 1.3 and older depots.

1 .2.5 struct ibp_timer

```
typedef  struct ibp_timer {
            int         ClientTimeout;
            int         serverSync;
} *IBP_timer;
```

The two timers have a completely different function. The **ClientTimeout** indicates the time the application is willing to wait for a response from the server. This parameter is used to improve the fault-tolerance of the IBP Client Library, to prevent waiting forever from an answer from a hanged server; or in case the network connection is particularly bad, or a very high latency time. The **ServerSync** is used as an ”or” condition: i.e., in a IBP_load operation on *IBP_FIFO* type of allocation, the application program can ask for N bytes or whatever gathered after **ServerSync** time. Another example is in a IBP_allocate(), if there is not enough of free space on IBP depot, IBP depot will hold the request for amount of time specified by **ServerSync** until either more space are available or timeout. This can be very helpful when another client is writing on the same media, and the application asking to load the data does not know how many bytes are written, but it’s willing to wait for some time before giving up.

2 IBP Allocate

```
IBP_set_of_caps   IBP_allocate (  IBP_depot              depot,
                                  IBP_timer              timeout,
                                  unsigned long int      size,
                                  IBP_attributes         attributes)
```

IBP_allocate() allocates a remote storage area on the host **depot** with specified resource. The allocated area has a maximum possible size of **size** bytes, and storage attributes, defined by **attributes**.

Return values

Upon success, **IBP_allocate**() returns an IBP_set_of_caps object. Otherwise, it 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 target host, invalid entry in **attributes**, ...)
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server running on target host.
- **IBP_E_INVALID_RID** : An invalid resource id.
- **IBP_E_UNKNOWN_RS** : Resource id can’t be found on IBP depot.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_WOULD_EXCEED_LIMIT** : Granting the request would cause the IBP server to exceed the maximum storage limit allocated to the storage category defined in **attributes**.
- **IBP_E_FILE_ACCESS** : 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_E_INTERNAL** : The IBP server has encountered an internal error while processing the client’s request.
- **IBP_E_TYPE_NOT_SUPPORTED** : A request to allocate a storage area of type IBP_FIFO was made to an IBP server that does not support this type.

3 IBP store

```
unsigned long int  IBP_store (  IBP_cap              cap,
                                IBP_timer            timeout,
                                 char                data,
                                 unsigned long int   size )
```

IBP_store() stores **size** bytes starting at **data** in the storage area accessed through the IBP capability **cap**. For this call to succeed, cap must be a write cap 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, IBP_CIRQ and IBP_FIFO. Data written to a storage area of type IBP_BUFFER overwrites any previous data (starting at the beginning of the buffer). If a ServerSync time is set, the call will return either if all the data has been written, or the time has expired.

Return values

Upon success, **IBP_store**() returns the number of bytes written. Otherwise it returns 0 and sets IBP_errno to one of the following error codes:

- **IBP_E_WRONG_CAP_FORMAT** : The IBP capability cap doesn't have the proper format.
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E_CAP_NOT_WRITE** : The IBP capability cap is not a write capability.
- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server running on target host.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_CAP_NOT_FOUND** : The storage area accessed through cap does not exist on the associated IBP server.
- **IBP_E_CAP_ACCESS_DENIED** : The storage area accessed through cap cannot be accessed for write operations.
- **IBP_E_SIZE_EXCEEDS_LIMIT** : The write operation would cause the aggregate size of the storage area to exceed the maximum size specified in the IBP_store() call. This error is only relevant for storage areas of type IBP_BYTEARRAY.
- **IBP_E_FILE_ACCESS**: 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_E_FILE_WRITE:** The IBP server has encountered an error while attempting to store incoming data to the underlying storage area**.**
- **IBP_E_RSRC_UNAVAIL**: 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_E_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

4 IBP load

```
unsigned long int IBP_load ( IBP_cap              cap,
                             IBP_timer            timeout,
                             char                 *buf,
                             unsigned long int    size,
                             unsigned long int     offset )
```

IBP_load() reads up to **size** bytes, starting at **offset**, from the storage area accessed through the IBP capability **cap**, into the memory area pointed by **buf**. For storage areas of type IBP_FIFO and IBP_CIRQ, offset is ignored. For this call to succeed, cap must be read cap returned by an earlier call to **IBP_allocate**(), or imported from the client which made the **IBP_allocate**() call. **IBP_load**() is a blocking call that returns only when all required data is read, the ServerSync expires, or the read operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_load**() returns the number of bytes actually read, otherwise it returns 0 and sets IBP_errno to one of the following error codes:

- **IBP_E_INVALID_PARAMETER** : One or more of the parameters to the **IBP_load**() call has an invalid value(e.g. negative size, ...)
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E_WRONG_CAP_FORMAT** : The IBP capability cap doesn't have the proper format.
- **IBP_E_CAP_NOT_READ** : The IBP capability cap is not a read capability.
- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server running on target host.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_CAP_NOT_FOUND** : The storage area accessed through cap does not exist on the associated IBP server.
- **IBP_E_CAP_ACCESS_DENIED** : The storage area accessed through cap cannot be accessed for write operations.
- **IBP_E_FILE_ACCESS**: 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_E_FILE_READ**: The IBP server has encountered an error while attempting to read incoming data to the underlying storage area.
- **IBP_E_RSRC_UNAVAIL**: 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_E_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

5 IBP copy

```
unsigned long int  IBP_copy (  IBP_cap              source,
                               IBP_cap              target,
                               IBP_timer            sourceTimeout,
                               IBP_timer            targetTimeout,
                               unsigned long int    size,
                               unsigned long int    offset )
```

IBP_copy() copies up to **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 and IBP_CIRQ, **offset** is ignored. As in other read operation to an IBP_FIFO or IBP_CIRQ 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 read cap returned by an earlier call to **IBP_allocate**(), or imported from the client which made the **IBP_allocate**() call, and target must be a write cap returned by a similar call. **IBP_copy**() is a blocking call that returns only when all required data is read, the ServerSync expires, or the read operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_copy**() returns the number of bytes actually read, otherwise it returns 0 and sets IBP_errno to one of the following error codes:

- **IBP_E _INVALID_PARAMETER**: One or more of the parameters to the IBP_copy() call has an invalid value(e.g.negative size, ...)
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E__WRONG_CAP_FORMAT** : The IBP capability cap doesn't have the proper format.
- **IBP_E_CAP_NOT_WRITE** : The IBP capability source is not a write capability.
- **IBP_E_CAP_NOT_READ** : The IBP capability target is not a read capability.
- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server running on target host.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_CAP_NOT_FOUND** : One (or both) storage area accessed through cap does not exist on the associated IBP server.
- **IBP_E_CAP_ACCESS_DENIED** : One (or both) storage area accessed through cap can not be accessed for write operations.
- **IBP_E_SIZE_EXCEEDS_LIMIT**: The write part of the copy operation would cause the aggregate size of the storage area to exceed the maximum size specified in the IBP_store() call. This error is only relevant for storage areas of type IBP_BYTEARRAY.
- **IBP_E_FILE_ACCESS**: 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_E_FILE_WRITE** : The IBP server has encountered an error while attempting to store incoming data to the underlying storage area.
- **IBP_E_FILE_READ** :The IBP server has encountered an error while attempting to read incoming data to the underlying storage area.
- **IBP_E_RSRC_UNAVAIL**: 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_E_INTERNAL** : The IBP server has encountered an internal error while processing the client's

6 IBP mcopy

```
Unsigned long int  IBP_mcopy ( IBP_cap              source,
                               IBP_cap              target[],
                               unsigned int         nCaps,
                               IBP_timer            srcTimeout,
                               IBP_timer            targetTimeout,
                               unsigned long int    size,
                               unsigned long int    offset,
                               int                  type[],
                               int                  port[],
                               int                  service )
```

IBP_mcopy() copies up to **size** bytes, starting at **offset**, from the storage area accessed through the IBP read capability **source**, and writes them to the storage area(s) accessed through the IBP write capability(ies) **target**[]. The number of target depots is stored in **nCaps**. For storage areas of type IBP_FIFO and IBP_CIRQ, 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 and IBP_CIRQ, 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 or IBP_CIRQ 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 read cap returned by an earlier call to IBP_allocate(), or imported from the client which made the IBP_allocate() call and target must be a write cap returned by a similar call. **IBP_mcopy**() is a blocking call that returns only when all required data is successfully copied from the source IBP serverto all the target IBP server, the highest ServerSync value expired or the operation is prematurely terminated due to an error.   At the moment, there are implemented five types of data movers that can support TCP , UDP and IP Multicast data transfer protocols, the selection of the protocol is passed through the API using the parameters **types**[] (at the moment same type should be specified for all the targets) and **service**, the first one specifies the data mover type for the target IBP servers, and the second specifies the type of data mover corresponding in the source IBP server; the dm **port**[] parameter is used to specify the data mover option, which at this point are just the ports to be used by the TCP and UDP protocols. It must be specified and it can be any port number higher than 1024. These types must be defined as follows:

	Targets	**Source**	**Description**
TCP	**DM_UNI**	**DM_SMULTI** **DM_MULTI** **DM_PMULTI**	**Sequential mcopy** **Round-robin fashion mcopy** **Threaded mcopy**
UDP	**DM_BLAST**	**DM_MBLAST**	**Threaded UDP Based mcopy**
MULTICAST	**DM_MCAST**	**DM_MULTICAST**	**Multicast mcopy**

Return values

Upon success, **IBP_mcopy**() returns the number of bytes read from the source capability, otherwise it returns 0 and sets IBP_errno to the same error codes as IBP_copy.

7 IBP manage

```
int  IBP_manage (  IBP_cap          cap,
                   IBP_timer        timeout,
                   int              cmd,
                   int              type,
                   IBP_CapStatus    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(except when **cmd** = IBP_PROBE, where any capability can be used). **cmd** can take one of the following values (defined in the file ”ibp_protocol.h”):

- **IBP_INCR** increments the reference count to the capability associated with the management capability cap, and whose type is specified in the parameter **type**. The parameter info is ignored for this command.
- **IBP_DECR** decrements the reference count to the capability associated with the management capability **cap** and whose type is specified in the parameter **type**. Decrementing the reference count the read capability associated with a storage area to 0 causes the IBP server to delete 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. The parameter info is ignored for this command.
- **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 storage area. Changing the size of a storage area of type IBP_FIFO or IBP_CIRQ 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.)

- **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.

type 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 structure of type IBP_CapStatus.

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

	type	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 errorcodes:

- **IBP_E_INVALID_PARAMETER** : One or more of the parameters to the **IBP_manage**() call has an invalid value (e.g. negative **type**, ...)
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E_WRONG_CAP_FORMAT** : The IBP capability doesn't have the proper format.
- **IBP_E_CAP_NOT_MANAGE** : The IBP capability is not a write capability.
- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_CAP_NOT_FOUND** : The storage area accessed through cap does not exist on the associated IBP server.
- **IBP_E_INVALID_MANAGE_CAP**: The management cap does not match the management cap associated with the storage area.
- **IBP_E_WOULD_DAMAGE_DATA** : Trying to change the size of a storage area of type IBP_ FIFO.
- **IBP_E_WOULD_EXCEED_LIMIT:** Trying to increase the maximum size of an IBP_BYTEARRAY type storage area leads exceeding the maximum storage space allocated for its class of storage.
- **IBP_E_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

8 IBP status

```
IBP_DptInfo  IBP_status ( IBP_depot             depot,
                          int                   StatusCmd,
                          IBP_timer             timeout,
                          Char                  *password,
                          unsigned long int     StableStor,
                          unsigned long int      VolStor
                          long                   duration );
```

IBP_status() allows an application to perform a query over a particular IBP depot resource and to modify some general storage properties. **depot** is the particular IBP depot the application would like to query. **StatusCmd** can have two values

- **IBP_ST_INQ** queries the IBP depot resource for its stable storage and the used amount, its volatile storage and the used amount, and the duration. When this command is used, the following 4 parameters are not used.
- **IBP_ST_CHANGE** changes the stable amount, the volatile amount and duration property of the IBP depot resource. Note the difference between this command and the IBP_manage() one. It is not possible to destroy the data already present in an IBP depot, so the changes only take effect if they are equal or bigger than the currently allocated area.

password is the IBP depot password. **StableStor** is the new total Stable Storage of the IBP depot. It must be equal or bigger than the current Stable Storage used; otherwise, it's ignored. **VolStor** is the new total Volatile Storage of the IBP depot. It must be equal or bigger than the current Volatile Storage used; otherwise, it's ignored. **duration** is the new maximum duration allowed. It is not retro-active.

Return values

Upon success, **IBP_status** () returns an IBP_DptInfo object, otherwise it returns a NULL and sets IBP_errno to one of the following error codes:

- **IBP_E_CONNECTION** : An error has occurred while trying to connect to the IBP server.
- **IBP_E_CLIENT_TIMEOUT** : A client timeout occurs.
- **IBP_E_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.
- **IBP_E_SOCK_READ** : An error has occurred while trying to read response from the IBP server.
- **IBP_E_INVALID_CMD** : The IBP server has received a command it does not recognize.
- **IBP_E_WOULD_DAMAGE_DATA** : Trying to change the size of a storage area of type IBP_ FIFO.
- **IBP_E_WOULD_EXCEED_LIMIT:** Trying to increase the maximum size of an IBP_BYTEARRAY type storage area leads exceeding the maximum storage space allocated for its class of storage.
- **IBP_E_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.