The Abstraction Layer

The Abstraction Layer (AL) is a library to interface a DTN application with the Bundle Protocol independently of the actual Bundle Protocol (BP) implementation. By decoupling the application code from the BP implementation, it is possible to reuse the same application code in different DTN environments, with significant advantages in terms of application portability, maintenance and interoperability. A possible drawback is the dependence of the AL on more than one BP implementation. At present the AL supports ION and DTN2 BP.s.
The AL consists of two elements:
- the AL Types;
- the AL APIs.

Note that as the present AL version has been created to support the DTNperf_3 application, only the Types and the APIs necessary for this purpose have been "abstracted". Other DTN applications could require the abstraction of others elements as well, thus extending the potentiality of the present version. The API documentation refers to October 2013.

Table of contents

Abstraction Layer Types....................1
Abstraction Layer APIs....................6
High Level APIs....................9
Abstraction Layer File and API structure....................13

Abstraction Layer Types

The AL Types are an abstraction of ION and DTN2 types. They are defined in the file "al_bp_types.h".
The types are divided into four groups: general types, registration EID types, bundle types, status report types.
In the table below is presented the correspondence between AL, DTN2 and ION types. If the cell is empty, there is not any correspondence.

Abstraction Layer	DTN2	ION
General Types		

al_bp_handle_t
int *
	dtn_handle_t
	int*
		BpSAP
		bpsap_st *{
		VEndpoint* vpoint;
		MetaEid endpointMetaEid;
		sm_SemId recvSemaphore; }

al_bp_endpoint_id_t {char uri[AL_BP_MAX_ENDPOINT_ID] }	dtn_endpoint_id_t {char uri[DTN_MAX_ENDPOINT_ID]}	char *

al_bp_timeval_t
u32_t
	dtn_timeval_t
	u_int
		DtnTime
		{
		unsigned long seconds;
		unsigned long nanosec;
		}

al_bp_timestamp_t	dtn_timestamp_t	BpTimestamp

{
u32_t secs;
u32_t seqno;
}
	{
	u_hyper secs;
	u_hyper seqno;
	}
		{
		unsigned long seconds;
		unsigned long count;
		}

al_bp_error_t
{ BP_SUCCESS
BP_ERRBASE;
BP_ENOBPI;
BP_EINVAL;
BP_ENULLPNTR;
BP_EUNREG;
BP_ECONNECT;
BP_ETIMEOUT;
BP_ESIZE;
BP_ENOTFOUND;
BP_EINTERNAL;
BP_EBUSY;
BP_ENOSPACE;
BP_ENOTIMPL;
BP_EATTACH;
BP_EBUILDEID
BP_EOPEN;
BP_EREG;
BP_EPARSEEID;
BP_ESEND;
BP_ERECV;
BP_ERECVINT;}		

Registration EID Types		
al_bp_reg_token_t u32_t	dtn_reg_token_t u_hyper	
al_bp_reg_id_t u32_t	dtn_reg_id_t u_int	

al_bp_reg_info_t
{ al_bp_endpoint_id_t endpoint;
al_bp_reg_id_t regid;
u32_t flags;
u32_t replay_flags;
al_bp_timeval_t expiration;
boolean_t init_passive;
al_bp_reg_token_t reg_token;
struct {
u32_t script_len;
char *script_val;} script;
}
	dtn_reg_info_t
	{dtn_endpoint_id_t endpoint;
	dtn_reg_id_t regid;
	u_int flags;
	u_int replay_flags;
	dtn_timeval_t expiration;
	bool_t init_passive;
	dtn_reg_token_t reg_token;
	struct {
	u_int script_len;
	char *script_val;} script;
	};	

al_bp_reg_flags_t
{BP_REG_DROP = 1,
BP_REG_DEFER = 2,
BP_REG_EXEC = 3,
BP_SESSION_CUSTODY = 4,
BP_SESSION_PUBLISH = 8,
BP_SESSION_SUBSCRIBE = 16,
BP_DELIVERY_ACKS = 32,}
	dtn_reg_flags_t
	{DTN_REG_DROP = 1,
	DTN_REG_DEFER = 2,
	DTN_REG_EXEC = 3,
	DTN_SESSION_CUSTODY = 4,
	DTN_SESSION_PUBLISH = 8,
	DTN_SESSION_SUBSCRIBE = 16,
	DTN_DELIVERY_ACKS = 32,}
		BpRecvRule
		{DiscardBundle,
		EnqueueBundle,
		}

Bundle Types		
al_bp_bundle_delivery_opts_t	dtn_bundle_delivery_opts_t	int

{
BP_DOPTS_NONE = 0,
BP_DOPTS_CUSTODY = 1,
BP_DOPTS_DELIVERY_RCPT = 2,
BP_DOPTS_RECEIVE_RCPT = 4,
BP_DOPTS_FORWARD_RCPT = 8,
BP_DOPTS_CUSTODY_RCPT = 16,
BP_DOPTS_DELETE_RCPT = 32,
BP_DOPTS_SINGLETON_DEST = 64,
BP_DOPTS_MULTINODE_DEST = 128,
BP_DOPTS_DO_NOT_FRAGMENT = 256,
}
	{
	DOPTS_NONE = 0,
	DOPTS_CUSTODY = 1,
	DOPTS_DELIVERY_RCPT = 2,
	DOPTS_RECEIVE_RCPT = 4,
	DOPTS_FORWARD_RCPT = 8,
	DOPTS_CUSTODY_RCPT = 16,
	DOPTS_DELETE_RCPT = 32,
	DOPTS_SINGLETON_DEST = 64,
	DOPTS_MULTINODE_DEST = 128,
	DOPTS_DO_NOT_FRAGMENT = 256,
	}
		{
		BP_DELIVERED_RPT;
		BP_RECEIVED_RPT;
		BP_FORWARDED_RPT;
		BP_CUSTODY_RPT;
		BP_DELETED_RPT;
		}

al_bp_bundle_priority_t
{
al_bp_bundle_priority_enum priority
{
BP_PRIORITY_BULK = 0,
BP_PRIORITY_NORMAL = 1,
BP_PRIORITY_EXPEDITED = 2,
BP_PRIORITY_RESERVED = 3,
}
u32_t ordinal;
}
	dtn_bundle_priority_t
	{
	COS_BULK = 0,
	COS_NORMAL = 1,
	COS_EXPEDITED = 2,
	COS_RESERVED = 3,
	}
		int
		{
		BP_BULK_PRIORITY (0)
		BP_STD_PRIORITY (1)
		BP_EXPEDITED_PRIORITY (2)
		}

al_bp_extension_block_t {
u32_t type;
u32_t flags;
struct {
u32_t data_len;
char *data_val;
} data;
}		

al_bp_bundle_spec_t
{
al_bp_endpoint_id_t source;
al_bp_endpoint_id_t dest;
al_bp_endpoint_id_t replyto;
al_bp_bundle_priority_t priority;
al_bp_bundle_delivery_opts_t dopts;
al_bp_timeval_t expiration;
al_bp_timestamp_t creation_ts;
al_bp_reg_id_t delivery_regid;
struct {
u32_t blocks_len;
	dtn_bundle_spec_t
	{
	dtn_endpoint_id_t source;
	dtn_endpoint_id_t dest;
	dtn_endpoint_id_t replyto;
	dtn_bundle_priority_t priority;
	int dopts;
	dtn_timeval_t expiration;
	dtn_timestamp_t creation_ts;
	dtn_reg_id_t delivery_regid;
	dtn_sequence_id_t sequence_id;
	dtn_sequence_id_t obsoletes_id;
	struct {	

al_bp_extension_block_t
*blocks_val;
} blocks;
struct {
u32_t metadata_len;
al_bp_extension_block_t
*metadata_val;
} metadata;
boolean_t unreliable;
boolean_t critical;
u32_t flow_label;
}
	u_int blocks_len;
	dtn_extension_block_t
	*blocks_val;
	} blocks;
	struct {
	u_int metadata_len;
	dtn_extension_block_t
	*metadata_val;
	} metadata;
	}	

al_bp_bundle_payload_locatio
n_t
{
BP_PAYLOAD_FILE = 0,
BP_PAYLOAD_MEM = 1,
BP_PAYLOAD_TEMP_FILE =
2,
}
	dtn_bundle_payload_location_
	t
	{
	DTN_PAYLOAD_FILE = 0,
	DTN_PAYLOAD_MEM = 1,
	DTN_PAYLOAD_TEMP_FILE
	= 2,
	}	

al_bp_bundle_id_t
{
al_bp_endpoint_id_t source;
al_bp_timestamp_t
creation_ts;
u32_t frag_offset;
u32_t orig_length;
}
	dtn_bundle_id_t
	{
	dtn_endpoint_id_t source;
	dtn_timestamp_t
	creation_ts;
	u_int frag_offset;
	u_int orig_length;
	}
		BundleId
		{
		EndpointId source;
		BpTimestamp creationTime;
		unsigned long
		fragmentOffset;
		}

al_bp_bundle_payload_t
{
al_bp_bundle_payload_loca
tion_t location;
struct
{
u32_t filename_len;
char *filename_val;
} filename;
struct
{
u32_t buf_len;
char *buf_val;
} buf;
al_bp_bundle_status_report
_t *status_report;
}
	dtn_bundle_payload_t
	{
	dtn_bundle_payload_locati
	on_t location;
	struct {
	u_int filename_len;
	char *filename_val;
	} filename;
	struct {
	u_int buf_len;
	char *buf_val;
	} buf;
	dtn_bundle_status_report_t
	*status_report;
	}
		Payload
		{
		unsigned long length;
		Object content;
		}

al_bp_bundle_object_t
{
al_bp_bundle_id_t * id;
al_bp_bundle_spec_t *
spec;
al_bp_bundle_payload_t *
payload;
}		

Status Report Types		

al_bp_status_report_reason_t
{
BP_SR_REASON_NO_ADDTL
_INFO = 0x00,
	dtn_status_report_reason_t
	{
	REASON_NO_ADDTL_INFO
	= 0x00,
		BpSrReason
		{
		
		SrLifetimeExpired = 1,

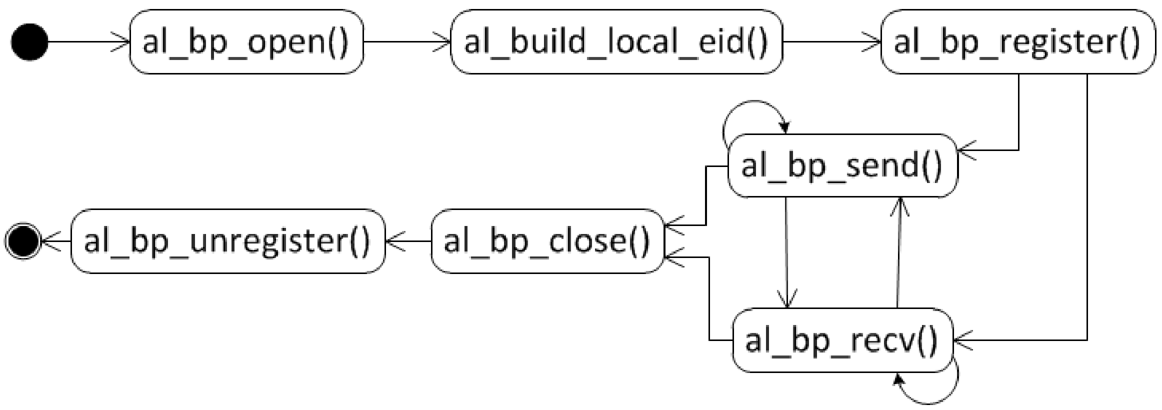
BP_SR_REASON_LIFETIME_EXPIRED = 0x01,
BP_SR_REASON_FORWARDED_UNIDIR_LINK = 0x02,
BP_SR_REASON_TRANSMISSION_CANCELLED = 0x03,
BP_SR_REASON_DEPLETED_STORAGE = 0x04,
BP_SR_REASON_ENDPOINT_ID_UNINTELLIGIBLE = 0x05,
BP_SR_REASON_NO_ROUTE_TO_DEST = 0x06,
BP_SR_REASON_NO_TIMELY_CONTACT = 0x07,
BP_SR_REASON_BLOCK_UNINTELLIGIBLE = 0x08,
}
	REASON_LIFETIME_EXPIRED = 0x01,
	REASON_FORWARDED_UNIDIR_LINK = 0x02,
	REASON_TRANSMISSION_CANCELLED = 0x03,
	REASON_DEPLETED_STORAGE = 0x04,
	REASON_ENDPOINT_ID_UNINTELLIGIBLE = 0x05,
	REASON_NO_ROUTE_TO_DEST = 0x06,
	REASON_NO_TIMELY_CONTACT = 0x07,
	REASON_BLOCK_UNINTELLIGIBLE = 0x08,
	}
		SrUnidirectionalLink,
		SrCanceled,
		SrDepletedStorage,
		SrDestinationUnintelligible,
		SrNoKnownRoute,
		SrNoTimelyContact,
		SrBlockUnintelligible
		}

al_bp_status_report_flags_t
{
BP_STATUS_RECEIVED = 0x01,
BP_STATUS_CUSTODY_ACCEPTED = 0x02,
BP_STATUS_FORWARDED = 0x04,
BP_STATUS_DELIVERED = 0x08,
BP_STATUS_DELETED = 0x10,
BP_STATUS_ACKED_BY_APP = 0x20,
}
	dtn_status_report_flags_t
	{
	STATUS_RECEIVED = 0x01,
	STATUS_CUSTODY_ACCEPTED = 0x02,
	STATUS_FORWARDED = 0x04,
	STATUS_DELIVERED = 0x08,
	STATUS_DELETED = 0x10,
	STATUS_ACKED_BY_APP = 0x20,
	}
		int
		{
		BP_STATUS_RECEIVE 0
		BP_STATUS_ACCEPT 1
		BP_STATUS_FORWARD 2
		BP_STATUS_DELIVER 3
		BP_STATUS_DELETE 4
		BP_STATUS_STATS 5
		}

al_bp_bundle_status_report_t
{
al_bp_bundle_id_t bundle_id;
al_bp_status_report_reason_t reason;
al_bp_status_report_flags_t flags;
al_bp_timestamp_t receipt_ts;
al_bp_timestamp_t custody_ts;
al_bp_timestamp_t forwarding_ts;
al_bp_timestamp_t delivery_ts;
al_bp_timestamp_t deletion_ts;
al_bp_timestamp_t ack_by_app_ts;
}
	dtn_bundle_status_report_t
	{
	dtn_bundle_id_t bundle_id;
	dtn_status_report_reason_t reason;
	dtn_status_report_flags_t flags;
	dtn_timestamp_t receipt_ts;
	dtn_timestamp_t custody_ts;
	dtn_timestamp_t forwarding_ts;
	dtn_timestamp_t delivery_ts;
	dtn_timestamp_t deletion_ts;
	dtn_timestamp_t ack_by_app_ts;
	}
		BpStatusRpt
		{
		BpTimestamp creationTime;
		unsigned long fragmentOffset;
		unsigned long fragmentLength;
		char *sourceEid;
		unsigned char isFragment;
		unsigned char flags;
		BpSrReason reasonCode;
		DtnTime receiptTime;
		DtnTime acceptanceTime;
		DtnTime forwardTime;
		DtnTime deliveryTime;
		DtnTime deletionTime;
		}

Abstraction Layer APIs

The AL APIs aims to decouple the application code from the API of a specific BP implementation. The scheme below summarizes the use of the most important AL APIs.



The AL APIs are defined in the file “al_bp_api.h”; every AL API calls the corresponding API of the specific BP implementation. APIs of DTN2 and ION are defined in the file “al_bp_dtn.h” and “al_bp_ion.h”.
The AL APIs are divided into three groups: principal APIs, utility APIs and high level APIs.
In the table below the correspondence between AL, DTN2 and ION APIs is presented for the principal APIs and utility APIs. High level APIs are not listed in the table because they do not correspond to any DTN2 or ION APIs. In fact, they have been designed to manage errors and to have a major control of the bundle as an object.

Abstraction Layer	DTN2	ION
Principal APIs		
al_bp_open(al_bp_handle_t* handle)	dtn_open(dtn_handle_t* handle)	bp_attach( )
al_bp_open_with_ip(char *daemon_api_IP, int daemon_api_port, al_bp_handle_t* handle)	dtn_open_with_IP(char *daemon_api_IP, int daemon_api_port, dtn_handle_t* handle)	
al_bp_errno(al_bp_handle_t handle)	dtn_errno(dtn_handle_t handle)	system_error_msg( )
al_bp_build_local_eid(al_bp_handle_t handle, al_bp_endpoint_id_t* const char* service_tag, char * type, char * eid_destination)	dtn_build_local_eid(dtn_handle_t handle, dtn_endpoint_id_t* local_eid, const char* service_tag)	
al_bp_register(al_bp_handle_t * handle, al_bp_reg_id_t* newregid)	dtn_register(dtn_handle_t handle, dtn_reg_info_t* reginfo, dtn_reg_id_t* newregid)	addEndpoint(char *endpointName, BpRecvRule recvAction, char *recvScript) bp_open(char * eid, BpSAP * ionptr)
al_bp_unregister(	dtn_unregister(	removeEndpoint(

al_bp_handle_t handle, al_bp_reg_id_t regid, al_bp_endpoint_id_t eid)	dtn_handle_t handle, dtn_reg_id_t regid)	char *endpointName)
al_bp_find_registration(al_bp_handle_t handle, al_bp_endpoint_id_t * eid, al_bp_reg_id_t * newregid)	dtn_find_registration(dtn_handle_t handle, dtn_endpoint_id_t* eid, dtn_reg_id_t* newregid)	findEndpoint(char *schemeName, char *nss, VScheme *vscheme, VEndpoint **vpoint, PsmAddress *elt)
al_bp_send(al_bp_handle_t handle, al_bp_reg_id_t regid, al_bp_bundle_spec_t* spec, al_bp_bundle_payload_t* payload, al_bp_bundle_id_t* id)	dtn_send(dtn_handle_t handle, dtn_reg_id_t regid, dtn_bundle_spec_t* spec, dtn_bundle_payload_t* payload, dtn_bundle_id_t* id)	bp_send(BpSAP sap, int mode, char * destEid, char * reportToEid, int lifespan, int classOfService, BpCustodySwitch custodySwitch, unsigned char srrFlags, int ackRequested, BpExtendedCOS* e-xtendedCOS, Object adu, Object *newBundle)
al_bp_recv(al_bp_handle_t handle, al_bp_bundle_spec_t* spec, al_bp_bundle_payload_location_t location, al_bp_bundle_payload_t* payload, al_bp_timeval_t timeout)	dtn_recv(dtn_handle_t handle, dtn_bundle_spec_t* spec, dtn_bundle_payload_location_t location, dtn_bundle_payload_t* payload, dtn_timeval_t timeout)	bp_receive(BpSAP sap, BpDelivery *dlvBuffer, int timeoutSeconds)
al_bp_close(al_bp_handle_t handle)	dtn_close(dtn_handle_t handle)	bp_close(BpSAP ionptr)
Utility APIs		
al_bp_get_implementation( )		
void al_bp_copy_eid(al_bp_endpoint_id_t* dst, al_bp_endpoint_id_t* src)	void dtn_copy_eid(dtn_endpoint_id_t* dst, dtn_endpoint_id_t* src)	
al_bp_error_t al_bp_parse_eid_string(al_bp_endpoint_id_t* eid, const char* str)	int dtn_parse_eid_string(dtn_endpoint_id_t* eid, const char* str)	int parseEidString(char *eidString, MetaEid *metaEid, VScheme **scheme, PsmAddress *schemeElt)
al_bp_error_t al_bp_get_none_endpoint(al_bp_endpoint_id_t * eid_none)		
al_bp_error_t al_bp_set_payload(al_bp_bundle_payload_t*	int dtn_set_payload(dtn_bundle_payload_t* payload,	

payload, al_bp_bundle_payload_location_t location, char* val, int len)	dtn_bundle_payload_location_t location, char* val, int len)	
void al_bp_free_payload(al_bp_bundle_payload_t* payload)	int dtn_free_payload(dtn_bundle_payload_t* payload)	zco_destroy_file_ref(Sdr sdr, Object fileRef)
void al_bp_free_extension_blocks(al_bp_bundle_spec_t* spec)		
void al_bp_free_metadata_blocks(al_bp_bundle_spec_t* spec)		
const char* al_bp_status_report_reason_to_str(al_bp_status_report_reason_t err)	const char* dtn_status_report_reason_to_str(dtn_status_report_reason_t err)	
al_bp_strerror char * al_bp_strerror(int err)		

Below we provide the reader with some basic information about the most important AL APIs, by pointing out the differences in case they run on top of DTN2 or ION BP implementations.

al_bp_open
al_bp_error_t al_bp_open(al_bp_handle_t handle)*

It opens the connection between the application and the BP daemon.
In DTN2 the API also initializes the handle.

al_bp_build_local_eid
al_bp_error_t al_bp_build_local_eid(al_bp_handle_t handle, al_bp_endpoint_id_t local_eid, const char* service_tag, char * type, char * eid_destination);*

It creates the local EID.
In DTN2 the local EID is taken from the handle.
In ION the local eid is constructed with specific rules dependent *type*'s value that can be: *Client, Server-CBHE, Monitor-CBHE, Server-DTN, Monitor-DTN.*

- *Client*: the local EID uses the same URI scheme as the destination;
  - if "ipn", the local EID will be **ipn:<own_number>:<own_pid>**
  - if "dtn" format the local eid will be **dtn://<local_hostname>/<service_tag>.**
- *Server-CBHE* or *Monitor-CBHE*: the local eid will be **ipn:<own_number>:<service_tag>;** the parameter *service_tag* is converted in integer.
- *Server-DTN* or *Monitor-DTN*: the local eid will be **dtn://<local_hostname>/<service_tag>.**

al_bp_register

*al_bp_error_t al_bp_register(al_bp_handle_t * handle, al_bp_reg_info_t* reginfo, _bp_reg_id_t* newregid)*

It registers the local EID to the BP daemon. In ION it also calls the API *bp_open()* that initializes the handle and allows the application to start sending and receiving bundles.

High Level APIs

High Level APIs aim to manage the bundle as an object with “get” and “set” APIs for almost all the bundle parameters .

al_bp_bundle_send

*al_bp_error_t al_bp_bundle_send(al_bp_handle_t handle, al_bp_reg_id_t regid, al_bp_bundle_object_t * bundle_object)*

It sends the bundle object.

al_bp_bundle_receive

al_bp_error_t al_bp_bundle_receive(al_bp_handle_t handle, al_bp_bundle_object_t bundle_object, al_bp_bundle_payload_location_t payload_location, al_bp_timeval_t timeout)

It receives the bundle object.

al_bp_bundle_create

*al_bp_error_t al_bp_bundle_create(al_bp_bundle_object_t * bundle_object)*

It creates an empty bundle object.

bp_bundle_free

*al_bp_error_t al_bp_bundle_free(al_bp_bundle_object_t * bundle_object)*

It deletes the bundle object from the memory.

al_bp_bundle_get_id

*al_bp_error_t al_bp_bundle_get_id(al_bp_bundle_object_t bundle_object, al_bp_bundle_id_t ** bundle_id)*

It takes the bundle Id from the bundle object.

al_bp_bundle_set_payload_location

*al_bp_error_t al_bp_bundle_set_payload_location(al_bp_bundle_object_t * bundle_object, al_bp_bundle_payload_location_t location)*

It sets the bundle payload location: either memory or file.

al_bp_bundle_get_payload_location

*al_bp_error_t al_bp_bundle_get_payload_location(al_bp_bundle_object_t bundle_object, al_bp_bundle_payload_location_t * location)*

It takes the bundle payload location.

al_bp_bundle_get_payload_size

*al_bp_error_t al_bp_bundle_get_payload_size(al_bp_bundle_object_t bundle_object, u32_t * size)*

It takes the bundle payload size.

al_bp_bundle_get_payload_file

*al_bp_error_t al_bp_bundle_get_payload_file(al_bp_bundle_object_t bundle_object, char_t ** filename, u32_t * filename_len)*

It takes the value of the payload if it is saved in a file.

bp_bundle_get_payload_mem

*al_bp_error_t al_bp_bundle_get_payload_mem(al_bp_bundle_object_t bundle_object, char ** buf, u32_t * buf_len)*

It takes the value of the payload if it is saved in the memory.

al_bp_bundle_set_payload_file

*al_bp_error_t al_bp_bundle_set_payload_file(al_bp_bundle_object_t * bundle_object, char_t * filename, u32_t filename_len)*

It sets the value of the payload if it is saved in a file.

al_bp_bundle_set_payload_mem

*al_bp_error_t al_bp_bundle_set_payload_mem(al_bp_bundle_object_t * bundle_object, * buf, u32_t buf_len)*

It sets the value of the payload if it is saved in the memory.

al_bp_bundle_get_source

*al_bp_error_t al_bp_bundle_get_source(al_bp_bundle_object_t bundle_object, al_bp_endpoint_id_t * source)*

It takes the bundle EID source.

al_bp_bundle_set_source

*al_bp_error_t al_bp_bundle_set_source(al_bp_bundle_object_t * bundle_object, al_bp_endpoint_id_t source)*

It sets the bundle EID source.

al_bp_bundle_get_dest

*al_bp_error_t al_bp_bundle_get_dest(al_bp_bundle_object_t bundle_object, al_bp_endpoint_id_t * dest)*

It takes the bundle EID destination.

al_bp_bundle_set_dest

*al_bp_error_t al_bp_bundle_set_dest(al_bp_bundle_object_t * bundle_object, al_bp_endpoint_id_t dest)*

It sets the bundle EID destination.

al_bp_bundle_get_replyto

*al_bp_error_t al_bp_bundle_get_replyto(al_bp_bundle_object_t bundle_object, al_bp_endpoint_id_t * replyto)*

It takes the status report EID destination

al_bp_bundle_set_replyto

*al_bp_error_t al_bp_bundle_set_replyto(al_bp_bundle_object_t * bundle_object, al_bp_endpoint_id_t replyto)*

It sets the status report EID destination

al_bp_bundle_get_priority

*al_bp_error_t al_bp_bundle_get_priority(al_bp_bundle_object_t bundle_object, al_bp_bundle_priority_t * priority)*

It takes the bundle priority.

al_bp_bundle_set_priority

*al_bp_error_t al_bp_bundle_set_priority(al_bp_bundle_object_t * bundle_object, al_bp_bundle_priority_t priority)*

It sets the bundle priority.

al_bp_bundle_get_expiration

*al_bp_error_t al_bp_bundle_get_expiration(al_bp_bundle_object_t bundle_object, al_bp_timeval_t * exp)*

It takes the bundle expiration time.

al_bp_bundle_set_expiration

*al_bp_error_t al_bp_bundle_set_expiration(al_bp_bundle_object_t * bundle_object, al_bp_timeval_t exp)*

It sets the bundle expiration time.

al_bp_bundle_get_creation_timestamp

*al_bp_error_t al_bp_bundle_get_creation_timestamp(al_bp_bundle_object_t bundle_object, al_bp_timestamp_t * ts)*

It takes the bundle creation timestamp.

al_bp_bundle_set_creation_timestamp

*al_bp_error_t al_bp_bundle_set_creation_timestamp(al_bp_bundle_object_t * bundle_object, al_bp_timestamp_t ts)*

It sets the bundle creation timestamp.

al_bp_bundle_get_delivery_opts

*al_bp_error_t al_bp_bundle_get_delivery_opts(al_bp_bundle_object_t bundle_object, al_bp_bundle_delivery_opts_t * dopts)*

It takes the bundle delivery options.

al_bp_bundle_set_delivery_opts

*al_bp_error_t al_bp_bundle_set_delivery_opts(al_bp_bundle_object_t * bundle_object, al_bp_bundle_delivery_opts_t dopts)*

It sets the bundle delivery options.

al_bp_bundle_get_status_report

*al_bp_error_t al_bp_bundle_get_status_report(al_bp_bundle_object_t bundle_object, al_bp_bundle_status_report_t ** status_report)*

It takes the bundle status report.

Abstraction Layer File and API structure

The typical directory structure is:
dtnperf/al_bp/src: declaration files + al_api.c
dtnperf/al_bp/src/bp_implementations: interfaces to either ION or DTN2.
From the application, which uses the al_bp APIs, to the APIs provided by the specific BP implementation we have a chain of intermediate calls.

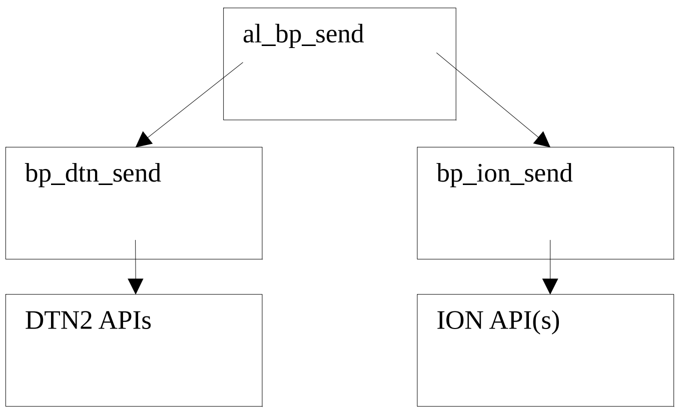
Let us explain this with an example, referring to al_bp_send.

al_bp_send
It is in al_bp_api.c,. It is called by the application. It just contains a switch to the BP implementation(s).

- DTN2
  bp_dtn_send (in al_bp_dtn.c). To avoid compilation errors, there is both a real implementation, if al_bp_is compiled for DTN2 or for DTN2&ION, and a dummy one at the file end, if al_bp is compiled for ION only.
  bp_dtn_send, in turns, call the bp DTN2 API(s).
- ION
  bp_ion_send (in al_bp_ion.c). To avoid compilation errors, there is both a real implementation, if al_bp_is compiled for ION or for DTN2&ION, and a dummy one at the file end, if al_bp is compiled for DTN2 only.
  bp_ion_send, in turns, call the bp ION API(s).

Types conversion are in files al_bp_dtn_conversions.c and al_bp_ion_conversions.c.

The prefix al_ion means the function takes a bp abstract type in and returns a ion type
so the conversion is bp -> ion
The prefix ion_al means the function takes a ion type in and returns a bp abstract type
so the conversion is ion -> bp


al_bp_send
bp_dtn_send
bp_ion_send
DTN2 APIs
ION API(s)