Coordinated Infrastructure for Fault Tolerant Systems

Fault Tolerance Backplane (FTB) API

FTB-Enabled Software Developer's Guide

Revision: FTB API Version 0.5 - Document Draft version 0.2

Contents

Chapter 1

Who should use this guide?

This guide is intended for users who wish to develop Fault Tolerance Backplance(FTB)-Enabled softwares. This developers guide mostly discusses the FTB API, using which FTB-enabled softwares can communicate (i.e publish and subscribe) fault-related information to the FTB, as well as other FTB-enabled softwares

Chapter 2

Terms and Conventions

This chapter explains some of the terms and conventions used in this guide.

Terms	Description
Component	A component is broadly defined as a piece of software or a software package. Examples include: MPICH2, Linux, PVFS, LAMMPS application, IB network library etc.
Component category	Components are logically separated into component categories for a systematic representation in the FTB system. Examples of component categories and their associated components include: MPI (with components like MPICH2, MVAPICH2, Open MPI), Applications (with components like NWCHEM, LAMPPS, SWIM) etc.
FTB Client	FTB Client is an entity that uses the FTB framework to exchange fault-related information. An example of an FTB client can be a FTB-enabled software piece in a process. For ex: A single process can contain code which is a part of the operating system component, MPI component and user application component. If all the 3 software pieces are FTB enabled, then each of them will constitute an FTB client. Each FTB client can connect to the FTB and send/receive fault information.

Region	A region is the first level of the FTB namespace. The region name 'ftb' is reserved by the FTB system. All component categories and components and event_names under the 'ftb' region name are also reserved by the CIFTS group. Semantics of event names (later described) are pre-established and understood for all events names in all the components and component categories in the 'ftb' region. For all other regions, the semantics are not defined and no component and components categories are reserved.
Event Name	An event name is a string that provides information about the fault. Within the 'ftb' region, event names are semantically pre-defined and understood for all the components and component categories. Event names are unique for a component and component category combination within the 'ftb' region. Examples of event name string formats: MPICH_ABORT, JOB_KILLED. An event name string can be composed of case-insensitive alphanumeric characters and underscores only.
Event severity	Event severity provides additional information about an event. The event severity is associated with the event name. An event name can have only one event severity. Currently, event severities are predefined by the FTB system.
Events	In the FTB framework, an event is an set of information. In reserved regions like 'ftb', an event can be uniquely identified by a combination of the component category, component and the event name. Associated with every event name is the predefined severity of that event. An FTB client can publish an event in an event namespace. Other FTB clients can subscribe with the FTB system to receive this event. The FTB framework is responsible for delivering the events between the different FTB clients.
Subscription String	This is a string (composed of wild-card options and specific attribute values) using which the FTB client specifies what events it wants to receive or subscribe to, as a part of its subscription.
Event Namespace	An Event namespace describes the space where a FTB client can throw an event, such that it is interpreted with the correct semantics by other FTB clients. The current FTB framework defines the namespace as a string of three parameters: 1. region_name, 2.component_category name 3.component_name.

Chapter 3

An Overview of CIFTS

The Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) project aims to provide an environment and infrastructure for sharing fault-related information, in order to help enable faults to be handled in a co-ordinated and holistic manner in the entire system. The Fault Tolerance Backplane (FTB) forms the back-bone of this CIFTS environment. FTB provides an infrastructure which can be used by different software in the system to exchange any fault-related information. The FTB also exposes an interface that can be used by different software to communicate and tie to the FTB.

The software in a high-end system that can potentially utilize the capabilities of FTB span operating systems, job schedulers, resource managers, middleware libraries, math libraries, file systems, applications, networking software etc.

3.1 The FTB Client Interface

The FTB Software has a layered architecture. This guide will not delve into the details of the internal FTB layers. The uppermost layer of FTB called FTB Client Interface is the most important layer from the FTB end-users perspective. This FTB Client Interface provides an API (Application Programming Interface) that should be used by any software wishing to communicate fault-related information with other software on the system using the FTB framework.

The rest of the guide gives details of the FTB Client Interface.

Chapter 4

Fault Tolerance Backplane (FTB) Client Interface

This chapter describes routines that are a part of the FTB Client Interface. Note that the various string lengths for arguments or their sub-fields can be found in Table 4.1

4.1 Connect to the FTB

```
int FTB_Connect
(
        IN const FTB_client_t *client_info
        OUT FTB_client_handle_t *client_handle
)
```

ARGUMENTS:

client_info: This structure provides information about the FTB client. Refer to Tables 4.2 and 4.3 for details of this structure.

client_handle: An opaque handle returned by the FTB system.

RETURNS:

FTB_SUCCESS: Indicates that client has successfully registered with the FTB system

FTB_ERR_EVENTSPACE_FORMAT: Indicates that user specified event_space field (part of the client_info structure) is not of required format

FTB_ERR_SUBSCRIPTION_STYLE: Indicates that the subscription style string has a different value than the ones permitted by FTB

FTB_ERR_INVALID_VALUE: Indicates that one of the fields in the client_info structure is invalid

FTB_ERR_DUP_CALL: Indicates that the client has already been registered and FTB_Connect is being called again

FTB_ERR_NULL_POINTER: Indicates that client_handle is a NULL pointer. User needs to pass a pointer pointing to a valid location

FTB_ERR_NOT_SUPPORTED: Indicates that the subscription_style is not supported. This error code is returned especially when an unsupported subscription style is used by a component on an architecture that cant support it. For example: subscription style of "FTB_subscription_notify" is not supported on IBM Blue Gene machines

DESCRIPTION:

This routine is to be used by every FTB client to initialize itself and connect to the FTB system. This is the first routine to be called by an FTB client and it can be called only once. The routine returns an opaque handle that will be used by the client during subsequent calls to identify itself.

For multi-threaded clients, this routine should be called only once. Different threads of the same process (i.e having same pid) cannot individually call this routine. It is up to the user to ensure that the FTB_Connect routine is the first FTB routine to be called by the process. Ideally, the main process should call this routine before threads get created.

It is possible, however, to trick the FTB system into believing that each thread is a **different client** if each thread in a process identifies itself with a unique client_name and then calls the FTB_Connect routine. This usefulness of this option is debatable and it needs to be throughly tested.

NOTE:

This routine was called FTB_Init in prior implementations.

4.2 Declare publishable events

```
int FTB_Declare_publishable_events
(
        IN FTB_client_handle_t client_handle
        IN const char *schema_file
        IN const FTB_event_info_t *event_info
        IN int num_events
)
```

ARGUMENTS:

client_handle: This is a opaque handle that was returned by the FTB system during the FTB_Connect call

schema_file: This is a string which indicates the absolute path and filename of the schema file. Setting the value to NULL indicates that the events are specified in the FTB client code through the event_info structure array (the third argument to this routine) and the num_events (the fourth argument to this routine). The working of schema files are indicated in Section 5.

event_info: A data structure containing information about the publishable events. Refer to Table 4.5 for additional details of this event_info data structure. This argument is ignored if a schema file is used to declare events

num_events: An integer specifying the number of events in the event_info array that the client wants to declare to FTB. This argument is ignored if a schema file is used to declare events

RETURNS:

FTB_SUCCESS: Indicates success

FTB_ERR_INVALID_HANDLE: Indicates an invalid client handle

FTB_ERR_INVALID_FIELD: Indicates that one of the fields (event name or severity) in the data structure event_info or the schema file is invalid

FTB_ERR_DUP_CALL: Indicates that this routine is being called more than once

FTB_ERR_DUP_EVENT: Indicates that the schema file or event_info structure contains a duplicate event. Event names within an event space should be unique

FTB_ERR_INVALID_SCHEMA_FILE: Indicates that the schema file may not be valid. This may include issues like: Schema file is not present, schema file does not have the correct read permissions, schema file does not

contain the correct event_space or the schema file is not of the right format

DESCRIPTION:

This routine will be called by the client to declare the events it plans to publish in its lifetime. This routine should be called before the client tries to publish any event using the FTB_Publish routine. This routine can be called only once - which means that all the events should be declared right in the beginning before the publishing can take place.

If the schema_file argument is not NULL, then its value will be treated as an absolute path to the schema file name. The routine will return an error code if the file is inaccessible or is of incorrect format. The event_info and num_events arguments are ignored in this case.

If the schema_file parameter is set to NULL, the event_info and num_events arguments will be considered. If num_events is set to 0, then no events will be registered (and event_info thus ignored) but the routine will however return FTB_SUCCESS.

For multi-threaded clients, the user should ensure that the routine gets called after FTB_Connect and before any FTB_Publish routine. Ideally, the the main process should call this routine before threads get created.

4.3 Publish events

```
int FTB_Publish
(
        IN FTB_client_handle_t client_handle
        IN const char *event_name
        IN const FTB_event_properties *event_properties
        OUT FTB_event_handle_t *event_handle
)
```

ARGUMENTS:

client_handle: This is the opaque handle that was returned by the FTB system during the FTB_Connect call

event_name: A case-insensitive string of size FTB_MAX_EVENT_NAME characters. The string can only consist of case-insensitive alphanumeric characters and the underscore character. The event_name should have been declared before this routine is called by calling the FTB_Declare_publishable_events routine

event_properties: The event_properties data structure is defined in Table 4.6

event_handle: A opaque handle that uniquely identify this published event

RETURNS:

FTB_SUCCESS: Indicates success

FTB_ERR_INVALID_EVENT_NAME: Indicates that the event name is invalid. It has either not been declared using the FTB_Declare_publishable_events routine

FTB_ERR_INVALID_EVENT_TYPE: Indicates that the user entered an invalid event_type in the event_properties data structure. The event_type is '1' for normal events (default if event_properties is NULL) and '2' for response events. Any other value explicitly specified by the user will return this error code

FTB_ERR_INVALID_HANDLE: Indicates an invalid client handle

FTB_ERR_NULL_POINTER: Indicates that the event handle pointer is NULL. This pointer should point to a valid memory location

DESCRIPTION:

This routine will be called by the client to publish events using event_name. The event_payload field (part of event_properties structure) will not be interpreted by FTB. The sender and the receiver should be in sync regarding the syntax and semantics of the payload.

For multi-threading clients, any thread can call this routine. The user needs to ensure that the event has been declared using the FTB_Declare_publishable_events routine before this FTB_Publish routine is called.

4.4 Subscribe to the FTB

```
int FTB_Subscribe
(
        OUT FTB_subscribe_handle_t *subscribe_handle
        IN FTB_client_handle_t client_handle
        IN const char *subscription_str
        IN int (*callback)(OUT FTB_receive_event_t *, OUT void*)
        IN void *arg
)
```

ARGUMENTS:

subscribe_handle: This is a opaque handle returned by the FTB system, that uniquely identifies this subscription

client_handle: This is the opaque handle that was returned to the client during the FTB_Connect call

subscription_str: A string that specifies the options that a client can base its subscriptions on. Currently, the subscription_str is of the format **"attribute1=value1, attribute2=value2, attribute3=value3"**. The supported attributes and values are defined in Table 4.7. The subscription_str is case-insensitive. If an attribute is not present in the subscription string, it will default to the value 'all', unless faced with constraints arising due to sub-dependencies with other fields. The subscription_str can be set to "" to subscribe to all events. *Examples of subscription_str : To subscribe to all events of severity fatal: subscription_str="severity=fatal", To subscribe to events of severity fatal and jobid=1234: subscription_str="severity=fatal, jobid=1234"*.

Specifying the correct subscription string is the users responsibility. For ex: If the user specifies an "event_name = MPICH_ABORT, event_space=ftb.os.all", it may never obtain that event since the publisher, in the 'ftb' region, may throw the event in the ftb.mpi.mpich2 eventspace only. However, the FTB system in this case **will not** return any error during FTB_Subscribe. Another example of a subscription_str of "event_name = MPICH_ABORT, severity=info" might not result in the subscriber getting any events if event_name = MPICH_ABORT is of severity=fatal. In particular, while specifying event_name in subscription_str, it is best not to mention the severity field, and if mentioned then set it to 'all' or to the correct value of that event_name.

int (*callback)(): This is the notification callback/handler function that the client wishes to register to handle events matching the above subscription string. This argument is set to NULL, if the client wants to get events using the polling mechanism instead of the notification mechanism.

void *arg: These are the arguments that the client can pass to the callback function (third argument), if it wants. This argument is set to NULL if the client is using the polling mechanism.

RETURNS:

FTB_SUCCESS: Indicates that the subscription was posted successfully

FTB_ERR_NULL_POINTER: Indicates that subscribe_handle is NULL

FTB_ERR_INVALID_HANDLE: Indicates that the client_handle is invalid

FTB_ERR_FILTER_ATTR: Indicates that the attribute name used in the subscription_str is not a valid name

FTB_ERR_FILTER_VALUE: Indicates that a value for the attribute used in the subscription_str is not valid

FTB_ERR_EVENTSPACE_FORMAT: Indicates that value for the event_space field is of incorrect format

FTB_ERR_SUBSCRIPTION_STR: Indicates that the subscription string is of an invalid format. This error code is also returned if the same attribute is specified twice in the subscription string (ex: subscription_str="severity=info, severity=info"

FTB_ERR_NOT_SUPPORTED: Indicates that the subscription method (polling, notification, both, neither) being used was not specified by the client during the FTB_Connect routine

DESCRIPTION:

This routine is used by the client to subscribe for events. The client specifies two things while subscribing to the FTB network.

1. The subscription criteria - which it specifies in the subscription string.
2. The mechanism (polling or notification) to be used to receiv e the events matching the above subscription criteria in the subscription string.

During the FTB_Connect call, if the client has specified "FTB_SUBSCRIPTION_NOTIFY" as the value for the client_subscription_style (part of the client_info data structure), then it needs to specify the callback function details in this current routine.

During the FTB_Connect call, if the client has specified "FTB_SUBSCRIPTION_POLLING" as the value for the client_subscription_style (part of the client_info data structure), then it needs to specify NULL in the third and fourth arguments of the routine call.

During the FTB_Connect call, if the client has specified "FTB_SUBSCRIPTION_NONE" as the value for the client_subscription_style (part of the client_info data structure), then this routine should not be called at all.

If the client specified "FTB_SUBSCRIPTION_BOTH" during the FTB_Connect call, then either notification or polling mechanism can be used, as described above.

An event will be reported only once. An event may match many subscription strings. It may thus have options wherein it can be obtained by the client using polling or notification mechanisms. There may be multiple valid callback/handler functions that can be trigerred in the case of a match. If an event matches both polling and notification, the notification mechanism(s) will have precedence over polling. If multiple callback/handler functions can be called - then the callback function for the first matching subscription string will be trigerred.

An example of this is as follows: Consider the three FTB_Subscribe calls made by a client, in the below order, with the following options-

1. subscription_str="severity=fatal,jobid=1234 " and subscription_style="FTB_SUBSCRIPTION_POLLING"
2. subscription_str="severity=fatal " and subscription_style="FTB_SUBSCRIPTION_NOTIFY" with callback function as func_callback1.
3. subscription_str="" and subscription_style="FTB_SUBSCRIPTION_NOTIFY" with callback function as func_callback2.

An event with "severity=fatal" and "jobid=1234" should actually be a match against all the three subscription strings. However, on the event arrival, the event will be matched against the subscription strings in the notification subscribe_style list, in the order in which they were subscribed. In this case, the event matches against subscription_str="severity=fatal" and the func_callback1 callback function will be called.

The FTB_Subscribe routine returns the subscribe_handle that can be used by the client at later stages to unsubscribe the subscription string from the FTB system.

For multi-threading clients, the FTB_Subscribe routine can be called by any thread.

NOTE:

This routine now incorporates the functionality that was a part of the FTB_Create_mask routine in prior FTB implementations. The concept of 'mask' is replaced by the subscription string.

4.5 Un-subscribe a subscription from the FTB network

```
int FTB_Unsubscribe
(
        INOUT FTB_subscribe_handle_t *subscribe_handle
)
```

ARGUMENTS:

subscribe_handle: This is a opaque handle that was returned by the FTB system during the FTB_Subscribe call. In the FTB_Unsubscribe routine, FTB updates the handle to make it invalid for use in subsequent calls

RETURNS:

FTB_SUCCESS: Indicates that the subscription was un-subscribed successfully

FTB_ERR_INVALID_HANDLE: Indicates that the subscribe_handle is invalid

DESCRIPTION:

This routine is used by the client to un-subscribe subscriptions from the FTB system. Once an subscription is un-subscribed, the client will no longer receive events matching that subscription string. The thread for the notification callback handler will be terminated.

For multi-threading clients, any thread can call the FTB_Unsubscribe routine. Since the FTB_Unsubscribe message may take some time to propagate in the FTB framework, the FTB agents may still forward some events matching the subscription string to the FTB client. Such events may be silently dropped by the FTB client library linked to the FTB client.

The user also needs to ensure that certain FTB routines like FTB_Connect, FTB_Subscribe, FTB_Disconnect etc. are called appropriately before or after FTB_Unsubscribe. If threads changes the sequence in which these routines are called, it may result in un-predictable behavior.

4.6 Get an event from event queue using polling

```
int FTB_Poll_event
(
        IN FTB_subscribe_handle_t subscribe_handle
        OUT FTB_receive_event_t *receive_event
)
```

ARGUMENTS:

subscribe_handle: This is the opaque handle that was returned by the FTB system during the FTB_Subscribe call. The subscribe_handle internally also indicates to the FTB the subscription string, matching which the event needs to be returned.

receive_event: This is a data structure containing information about the received event. Refer to Table 4.8 for details on the receive_event data structure.

RETURNS:

FTB_SUCCESS: Indicates an event was successfully obtained from queue

FTB_ERR_NULL_POINTER: Indicates that receive_event is a NULL pointer. This pointer should point to a valid memory location

FTB_ERR_INVALID_HANDLE: Indicates that the subscribe_handle is invalid

FTB_ERR_NOT_SUPPORTED: Indicates that the polling mechanism is not a supported mechanism for this client and the provided subscribe_handle

FTB_GOT_NO_EVENT: Indicates no event was present in the queue

DESCRIPTION:

This routine is used by the client to check if there is any event matching a **particular subscription string** present in the queue. The client needs to provide the subscribe_handle obtained from the FTB_Subscribe routine (that was called to subscribe that particular subscription string). If an event is successfully obtained from the queue, it will be returned in the receive_event data structure.

The receive_event data structure contains a field named event_type. This field is important from the Event association (refer to section 4.9) point-of-view. If event_type is '1', the received event is considered a *normal event*

and the interpretation of the event_payload field is left to the client. If event_type is '2', the received event is considered a *response or follow-up* to a prior published or received event. In this case, it is expected that the **received event payload** should have the **event_handle** of the prior event as the first field.

If a client wants to generate the event_handle for any received event, it can do so by calling the FTB_Get_event_handle routine described in the later sections of the guide. Please refer to Section 4.9 on 'Event association' to get a better understanding of event_types and event_handles.

For multi-threading clients, any thread can call this routine.

4.7 Disconnect the client from FTB

```
int FTB_Disconnect
(
        IN FTB_client_handle_t client_handle
)
```

ARGUMENTS:

client_handle: This is a opaque handle that was returned by the FTB system during the FTB_Connect call.

RETURNS:

FTB_SUCCESS: Indicates success

FTB_ERR_INVALID_HANDLE: Indicates invalid client handle

DESCRIPTION:

This routine will be used to disconnect the client from FTB. It will terminate all FTB-related existing connections and free-up all resources.

For multi-threaded clients, only one thread can call FTB Disconnect. It is ideally recommended that the main thread call FTB_Disconnect after all threads have terminated. It it up to the user to ensure that FTB_Disconnect is the last routine to be called for FTB.

4.8 Additional error codes

All the above routines may return some additional error codes, as follows. These error codes may reflect the internal state of FTB

1. FTB_ERR_NETWORK_GENERAL - An internal general network error
2. FTB_ERR_NETWORK_NO_ROUTE - The FTB system could not find a route to send the message
3. FTB_ERR_INVALID_PARAMETER - FTB unexpectedly failed on a require parameter

4.9 Associating events

In the FTB framework, on receiving an event, the FTB clients may frequently find a need to publish a *response* event. An FTB client may also find a need to publish a *follow-up* event to its prior published event.

'Event association' takes place when an event in published as a follow-up to a prior published event or as a response to a received event. Consider the below examples for usage scenarios when this may take place

1. A component may publish an "potential failure" event (event 1). After some time, it may want to publish a "recovered from failure" follow-up event (event 2). It will be useful if it can associate event 2 and indicate it as a follow-up event to event 1.
2. MPI publishes "cannot communic ate with node 1" event (event 1). The InfiniBand network library, then, publishes "communication re-established" event (event 2) and indicates that this is a response event to event 1.
3. OS publishes "process x has 100% cpu usage". The scheduler, then, publishes a response event "process x: priority lowered".

Event association will provide a mechanism to exchange some level of event-response information among different clients.

The current FTB implementation implements event association through the use of event handles. At the sender FTB client end, FTB_Publish() routine returns a unique event handle for every event it publishes. At the receiver FTB client end, the client can request for this event handle to be generated from the received message (see Table 4.8 for the received message data structure) using the FTB_Get_event_handle routine.

When the receiver FTB client wants to publish a response event (named event2) in response to a received event (named event1), it does the following

1. Obtain and keep track of the event_handle for event1. The event_handle for event1 can be obtained using the FTB_Get_event_handle routine (described in next section). The FTB_Get_event_handle routine re-generates the event_handle for event1 from the FTB_receive_event_t structure, which contains the received event1
2. Create the *event properties* structure to be passed to FTB_Publish routine for event2. In this event_properties structure, set the event_type to '2' and copy the event_handle for event1 in the event_payload section
3. Publish the event using FTB_Publish

When an FTB client wants to publish an event (named event2) as a follow-up to its prior published event (named event1), it does the following

1. Keep track of the event_handle provided to it on the return of the FTB_Publish routine for event1
2. Create the *event properties* structure to be passed to FTB_Publish routine for event2. In this event_properties structure, set the event_type to '2' and copy the event_handle for event1 in the event_payload section

When the new receivers receive the response/follow-up event, they should check the event_type field. If the field indicates that the event is a follow-up event, the new receiver client can read the original event's event_handle from the received event's payload section.

The Event association feature currently works with the following assumptions

1. Event handles are opaque to the FTB client
2. The FTB system does not maintain a record of published or received events. It is the client's responsibility to keep track of the published event handles and the received events. The client can use the FTB_Compare_event_handle routine to compare event handles to determine a match.
3. From the FTB systems point-of-view, the event_type and event_payload are transparent fields. The FTB system makes no decisions based on these fields. Thus, event association is transparent to FTB. The FTB system will not attempt to send the response/follow-up events to any specific destination
4. The subscription_string does not contain any specific criteria for subscribing to follow-up events. A FTB client can only realize that an event is a follow-up event after examining the event_type field in the received event

4.9.1 Get the event_handle

```
int FTB_Get_event_handle
(
        IN const FTB_receive_event_t receive_event
        OUT FTB_event_handle_t * event_handle
)
```

ARGUMENTS:

event_handle: This is a opaque handle that identifies the event that was published by the FTB client

RETURNS:

FTB_SUCCESS: Indicates that event handle was successfully returned

FTB_FAILURE: Indicates that event handle could not be generated for some reason

DESCRIPTION:

This routine will return an event handle from a receive_event structure. The event handle will be an opaque handle.

For multi-threaded clients, any thread can call this routine

4.9.2 Compare event_handles

```
int FTB_Compare_event_handles
(
        IN const FTB_event_handle_t event_handle1
        IN const FTB_event_handle_t event_handle2
)
```

ARGUMENTS:

event_handle: This is a opaque handle that identifies the event that was published by the FTB client

RETURNS:

FTB_SUCCESS: Indicates that handles match

FTB_FAILURE: Indicates that handles do not match

FTB_ERR_INVALID_HANDLE: Indicates invalid event handle

DESCRIPTION:

This routine can be used by the FTB client to compare two event handles. This would most likely be useful when the received event has an event_type of '2', whose payload contains an event_handle of the original event.

For multi-threaded clients, any thread can call this routine

Table 4.1: Maximum values for FTB_MAX_*

Field Name	**Value**
FTB_MAX_CLIENTSCHEMA_VER	8
FTB_MAX_EVENTSPACE	64
FTB_MAX_CLIENT_NAME	16
FTB_MAX_CLIENT_JOBID	16
FTB_MAX_EVENT_NAME	32
FTB_MAX_SEVERITY	16
FTB_MAX_HOST_NAME	64
FTB_MAX_PID_STARTTIME	32
FTB_MAX_PAYLOAD_DATA	368

Table 4.2: Data Structure client_info

Field Name	**Field Description**
char client_schema_ver [FTB_MAX_CLIENTSCHEMA_VER]	This is a string of length FTB_MAX_CLIENTSCHEMA_VER, including the terminating null character. This field is **reserved** for now. In the future, it will be used to specify the component schema version that the client is conforming to, in its implementation.
char event_space [FTB_MAX_EVENTSPACE]	The event_space field identifies the namespace in which the client will publish events in its lifetime. event_space is a string of length FTB_MAX_EVENTSPACE (including the terminating null character) of the format region_name.component_category.component_name. Details of the event_space string split-up are as follows. 1. region_name is a character sequence of alphanumeric and underscore characters. 'region' is the first-level hierarchy of the FTB namespace. The region_name of 'ftb' is considered as reserved for all CIFTS-recognized component categories and component names. A component publishing an event in the **'ftb'** region of the namespace will have had the semantic behavior of its publishable events well-defined in the public domain. 2. component_category is a character sequence of alphanumeric and underscore characters. It refers to the category a component belongs to. Examples include filesystems, os, mpi, applications etc. Component categories in the 'ftb' region are assigned and provided by the CIFTS group only. 3. component_name is a character sequence of alphanumeric and underscore characters. It refers to the name of the component. Examples: For the component_category= 'mpi' - components names may include mpich2, mvapich2, openmpi etc. Component names and component categories for the 'ftb' region are assigned by the CIFTS group. The event_space string is a mandatory field provided by the user. There is NO DEFAULT value assigned to it. The region_name, component_category, component_name character sequences can consist of alphanumeric and the underscore character(s) only. Each of these three sequences are concatenated using a '.' to form the event_space string. The three fields region_name, component_category and component_name can be of any lengths as long as the entire event_space string (including the terminating null character) does not exceed FTB_MAX_EVENTSPACE. The list of component categories and component values reserved in the 'ftb' region, by the CIFTS group, can be found in Table 4.4.
char client_name [FTB_MAX_CLIENT_NAME]	This is a string of case insensitive alphanumeric and underscore characters, of length FTB_MAX_CLIENT_NAME, including the terminating null character. There is NO DEFAULT value.
char client_jobid [FTB_MAX_CLIENT_JOBID]	This field is set by the user to correspond to the Job id of the process. It is of size FTB_MAX_CLIENT_JOBID characters, including the terminating null character. There is NO DEFAULT value. ……..continued in Table 4.3

Table 4.3: Data Structure client_info ... continued

Field Name	Field Description
char client_subscription_style [32]	This field indicates what mechanisms will be supported by this client to get the events that it will subscribe for during its lifetime. The following string values are available for this field. 1. "FTB_SUBSCRIPTION_POLLING" - Client will make an explicit call and get the event from its event queue. The event queue will be populated by the FTB system based on client's subscription criteria. The "FTB_SUBSCRIPTION_POLLING" option indicates that the client plans to receive events by the polling mechanism only. It will not be allowed to used notification mechanisms. 2. "FTB_SUBSCRIPTION_NOTIFY" - Client will register a notification callback/handler function for its subscriptions. The callback function will be called by the FTB library on a match between the incoming event and subscription criteria. The "FTB_SUBSCRIPTION_NOTIFY" option indicates that the client plans to receive events by the notification mechanism only. It wont be allowed to use polling mechanism. 3. "FTB_SUBSCRIPTION_NONE" - Client plans to not subscribe to any events in its lifetime and plans to only publish events. This helps FTB avoid unnecessary resource allocation. 4. "FTB_SUBSCRIPTION_BOTH" - Client plans to get events by using both polling and notification callback/handler mechanisms. There is NO DEFAULT value automatically assigned to this field. The error code FTB_ERR_SUBSCRIPTION_STYLE is returned if a different value, other than the ones discussed above is assigned to the client_subscription_style variable. For BGL systems, the "FTB_SUBSCRIPTION_NOTIFY" and the "FTB_SUBSCRIPTION_BOTH" options are not supported on BGL and if these values are specified, a error code of FTB_ERR_NOT_SUPPORTED is returned.
unsigned int client_polling_queue_len	The client can use this parameter to set the size of the polling queue. This parameter will only be considered if the FTB_SUBSCRIPTION_POLLING or FTB_SUBSCRIPTION_BOTH values are specified in the client_subscription_style field. The default value of this field is specified by the FTB_DEFAULT_POLLING_Q_LEN (currently set to 64) parameter in FTB. The default value is used if the value of the client_polling_queue_len is set as less than or equal to 0 by the user. The FTB_DEFAULT_POLLING_Q_LEN will become a tunable parameter in a future version of FTB.

Table 4.4: Known Components and Component Categories for the 'ftb' region(will be changed in coming months)

Component Category	Component Name
mpi	mpich2, mvapich2, openmpi, lammpi
filesystem	pvfs
os	linux, bgl-cnk
applications	swim, nwchem, lammps
networks	ib
rm_js	cobalt
checkpoint_sw	blcr
math_lib	ft_la
test1	test1

Table 4.5: Data Structure: event_info

Field Name	Field Description
char event_name[FTB_MAX_EVENT_NAME]	This is a case-insensitive string of alphanumeric and underscore characters, of FTB_MAX_EVENT_NAME characters, including the terminating null character.
char severity[FTB_MAX_SEVERITY]	This is a case-insensitive string of FTB_MAX_SEVERITY characters. The severity needs to be one of the following values (as defined in the FTB system): 'fatal', 'error', 'info', 'warning'

Table 4.6: Data Structure: event_properties

Field Name	Field Description
char event_type	event_type is an integer field which is reserved for now. It will have the following values: '1' for Normal events '2' for Response events. The values are pre-defined by the FTB system. If the event_properties data structure was set to NULL, then FTB will treat the event as event_type of '1'.
char event_payload[FTB_MAX_PAYLOAD_DATA]	This field contains the user-defined payload. The contents of this field cannot be interpreted by the FTB. The payload is currently limited to FTB_MAX_PAYLOAD_DATA bytes. If the event_type is '2', it is *expected* that the first entry in this field will be the event_handle of the event that the client in responding to. If the event_type is '1', the entry interpretation is left to the clients.

Table 4.7: Supported Attributes and Values for Criteria Strings

Attribute Name	**Possible Values**	**Type/Size of Value**
severity	'all', 'fatal', 'info', 'error', 'warning'	Predefined in the FTB system. Any other value will return the FTB_ERR_FILTER_VALUE error code.
event_space	Format: region_name.component_category.component_name. 'all' is an acceptable value of all the 3 sub-fields. The three sub fields of event_space field are composed of alphanumeric and underscore characters. The total length of event_space is defined by FTB_MAX_EVENTSPACE, currently set to 64 characters. An error in the format of the event_space will cause FTB to return the FTB_ERR_EVENTSPACE_FORMAT error code. In later FTB versions, the component_category and component_name will be cross-checked against pre-defined values for region_name, component_category and component. for the 'ftb' region_name.	
jobid	'all', any string	String with maximum length FTB_MAX_JOBID (16 characters).
host_name	'all', any string	String with maximum length FTB_MAX_HOST_NAME (64 characters)
event_name	'all', any string	String with maximum length FTB_MAX_EVENT_NAME (32 characters). Event name is a string of case-insensitive alphanumeric characters and the underscore character.
Empty string	-	Subscribe to all events

Table 4.8: Data Structure receive_event

Field Name	**Field Description**
char event_space[FTB_MAX_EVENTSPACE]	The event_space is a string of format region_name.component_category.component_name and length FTB_MAX_EVENTSPACE (which includes the terminating null character)
char event_name[FTB_MAX_EVENT_NAME]	The event_name string is of length FTB_MAX_EVENT_NAME (including the terminating null character)
char severity[FTB_MAX_SEVERITY]	This string specifies the severity of the event and is of length FTB_MAX_SEVERITY(including the terminating null character)
char client_jobid[FTB_MAX_JOBID]	This string specifies the Jobid and is of length FTB_MAX_JOBID (including the terminating null character)
char client_name[FTB_MAX_CLIENT_NAME]	This string of length FTB_MAX_CLIENT_NAME characters (including the terminating null character) is one of the fields that helps identify a FTB client
uint8_t client_extension	This is a field which identifies whether the sender FTB client is a IBM Blue Gene machine or not. This would not be required by the user usually, but would be required internally by FTB if the user wants to regenerate the event handle using the FTB_Get_event_handle() routine.
uint16_t seqnum	The sequence number for this event from the sender side.
FTB_location_id_t incoming_src	This data structure gives details of the src of the message. The data structure contains the following fields 1. char hostname[FTB_MAX_HOST_NAME] - A string of size FTB_MAX_HOST_NAME for the hostname 2. process_id_t process_id - The PID of the client process as obtained by FTB 3. char pid_starttime_t pid_starttime[FTB_MAX_PID_STARTTIME] - This start time of the process with process id PID
uint8_t event_type	This indicates the type of event. A value of '1' indicates the event is a normal event and the interpretation of the payload is left to the user. A value of '2' indicates that the user is a response event and that the payload should contain an event handle.

Chapter 5

Dealing with Schema files

Publishing events is an important aspect of the FTB. The FTB requires that an FTB-enabled software pre-declare the events (and attributes of these events) to the FTB prior to publishing them. Within the FTB framework, there are two ways to do it.

1. Compile time event declaration: Describe the events within the code of the software. These events are then passed as a parameter to the FTB_Declare_publishable_events routine. More information on this can be found in Chapter 4.
2. Run-time event declaration: Use the schema files, to read the events at run-time. The absolute path (including the filename) to the file needs to be passed as an argument to the FTB_Declare_publishable_events routine. For more information on the FTB_Declare_publishable_events routine, refer to Chapter 4.

This chapter deals with the format of the schema file.

5.1 Rules for the schema files

The schema file follows the below rules and guidelines.

1. Every client has its own specific schema file, at a location which is available during run-time
2. This location is indicated in the FTB_Declare_publishable_events routine call.
3. Comments in this file are preceded by # character
4. Blank lines are acceptable

5. The FTB library will being reading the data after the "start" keyword and stop reading once it encounters the "end" keyword
6. Following the "start" keyword, the event_space of the component is expected. Please read the preceding chapters to understand the compostion and semantics of the event_space.
7. Following the event_space, the event names and severity if the format *event_name, severity* are expected
8. FTB_Declare_publishable_events routine will return errors in the above rules are not followed

An example of the schema file is as follows:

```
start
region_name.component_category.component_name #This is the event_space
event1 name, severity
event2 name, severity
end
```

Chapter 6

Sample Software and Examples of the FTB API

This chapter explains a few example software to demonstrate the FTB API. Most of these examples are a part of the FTB source code, and can be found in its *c*omponents directory.

6.1 Example 1: Periodic Watchdog

The watchdog software is used to check the availability of the FTB backplane. It publishes events and waits to receive those events.

The software demonstrates the following

1. Declaring publishable events within the code
2. Subscribing to events using the polling mechanism
3. Declaring publishable events in a schema file
4. Subscribing to events using the callback mechanism

A simplified version of the FTB watchdog example code is presented in Listing 6.1. Step 2 of the below code shows how to declare the publishable events, in the code itself, using the FTB_Declare_Publishable_events routine.

Step 3 and Step 5 in Listing 6.1 also shows how to obtain subscribe for events using the polling mechanism. In particular, Step 5 shows how to use the FTB_Poll_event routine to pull the event from the event queue.

Listing 6.1: FTB Watchdog pseudocode

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include "libftb.h"

int main(int argc, char *argv[])
{
    FTB_client_t cinfo;
    FTB_client_handle_t handle;
    FTB_subscribe_handle_t shandle;
    int ret = 0; iter = 0;

    /**** STEP-1 ****/
    ret = FTB_Connect(&cinfo, &handle);
    if (ret != FTB_SUCCESS) {
            printf("FTB_Connect is not successful"); exit(-1);
    }

    /**** STEP-2 ****/
    FTB_event_info_t event_info[1] = { {"WATCH_DOG_EVENT", "INFO"} };
    ret = FTB_Declare_publishable_events(handle, 0, event_info, 1);
    if (ret != FTB_SUCCESS) {
            printf("FTB_Declare_Publishable_events failed"); exit(-1);
    }

    /**** STEP-3 ****/
    char * subscription_str = "event_space=ftb.all.watchdog";
    ret = FTB_Subscribe(&shandle, handle, subscription_str, NULL, NULL);
    if (ret != FTB_SUCCESS) {
```

```
            printf("FTB_Subscribe failed"); exit(-1);
    }

    while (iter != 10) {
            FTB_receive_event_t caught_event;
            FTB_event_handle_t ehandle;


        /**** STEP-4 ****/
            ret = FTB_Publish(handle, "WATCH_DOG_EVENT", NULL, &ehandle);
            if (ret != FTB_SUCCESS) {
                printf("FTB_Publish failed"); exit(-1);
            }

            sleep(1);


        /**** STEP-5 ****/
            ret = FTB_Poll_event(shandle, &caught_event);
            if (ret != FTB_SUCCESS) {
                printf("No event caught"); break;
        }

        iter++;
    }

    /**** STEP-6 ****/
    FTB_Disconnect(handle);
    return 0;
}
```

Listing 6.2 shows how to specify a schema file in the code using the FTB_Declare_Publishable_events routine. In code Listing 6.1, one can replace Step 2 by Listing 6.2 to achieve the same effect with the schema file. Please note that the schema file needs to be available at run-time

Listing 6.2: Specifying schema files in a code

```
/**** STEP-2 ****/
ret = FTB_Declare_publishable_events(handle, "watchdog_schema.ftb", NULL, 0);
if (ret != FTB_SUCCESS) {
        printf("FTB_Declare_Publishable_events failed ");  exit(-1);
}
```

Listing 6.3 shows the format of the schema file for the FTB Watchdog code.

Listing 6.3: Schema File for FTB Watchdog

```
start
ftb.ftb_examples.watchdog
watch_dog_event  info
end
```

Lastly, Listing 6.4 shows how to replace Step 3 and Step 5 in Listing 6.1 by Step3 in Listing 6.4 to subscribe to events using the notification function. Of course, the relevant notification function will also need to be provided, an example of which is given in Lsiting 6.5.

Listing 6.4: Subscribe to events using notification

```
/**** STEP-3 ****/
char * subscription_str = "event_space=ftb.all.watchdog";
ret = FTB_Subscribe(&shandle, handle, subscription_str, watchdog_receiver_func, 
if (ret != FTB_SUCCESS) {
        printf("FTB_Subscribe failed ");  exit(-1);
```

```
    }
```

Listing 6.5: Example notification callback handler

```
void watchdog_receiver_func(FTB_receive_event_t *caught_event)
{
    printf("Received event details: Event space=%s, Severity=%s,
            Event name=%s, Client name=%s,
            Hostname=%s, Seqnum=%d\n", caught_event.event_space,
                caught_event.severity, caught_event.event_name,
                caught_event.client_name, caught_event.incoming_src.hostname,
                caught_event.seqnum);

    return;
}
```

Further examples can be found in the FTB source code. Please refer to the README in the source code for more information.