Suspend/Resume handling in Technicolor DDS

Contents

Suspend/Resume support

In order to save on battery consumption, Technicolor DDS supports a suspend/resume mechanism to allow devices to go into a deep sleep and wake up afterwards with all DDS topics, i.e. DataWriters and DataReaders fully restored.

In addition to this useful feature, it is possible to use a Background Notification Service that can automatically wake up sleeping devices whenever data arrives on topics in which the sleeping devices are interested. To allow this feature, it is necessary to deploy a Background Notification Server somewhere in the same domain, which could be colocated with the DDS Forwarder.

Controlling DDS activities

The suspend/resume functionality is based on fine-grained control of internal DDS activities.

Following activities are defined (see `/dds/dds_aux.h` for the exact data definitions):

Table 1: DDS activities

Activity	Description
DDS_TIMER_ACTIVITY	Controls whether internal timers are running.
DDS_UDP_ACTIVITY	Whether the UDP/IP and UDP/IPv6 transport subsystems are active (sockets are present).
DDS_TCP_ACTIVITY	Whether the TCP/IP and TCP/IPv6 transport subsystems are active (sockets are present).
DDS_DEBUG_ACTIVITY	Whether the internal DDS debug shell stays active.

Note: An additional definition exists to specify all activities at once: DDS_ALL_ACTIVITY

Following functions are present to control these activities from within application programs:

```
void DDS_Activities_suspend (DDS_Activity activities);

void DDS_Activities_resume (DDS_Activity activities);
```

When an application needs to go to a deep sleep, it should call DDS_Activities_suspend (DDS_ALL_ACTIVITY) to force DDS into a frozen operating state. In this state, all DDS entities still exist, even the discovered participants are still present, but the low-level protocol sockets are closed, all timers are frozen and the debug shell is no longer working. The net result is that all datacaches stay populated with data and DDS can resume at any time.

When the application is ready to resume operation after waking up, it should call DDS_Activities_resume (DDS_ALL_ACTIVITY).

This will recreate all underlying sockets, and continue all active timers, adjusting them for the amount of time that has actually passed. As a result of this, a number of possible things might happen:

- Either the participant hasn't timed out yet, and operation will seemlessly resume, or
- The participant has timed out and all associations with it will be removed completely.

Either way, operation of DDS resumes and other participants can now be discovered again.

The net result of DDS being frozen leads to this participant becoming timed out in the participants it was talking to. Of course, once it resumes operation, all peers will rapidly become available again.

In practice, no UDP or TCP sockets stay open while frozen, so the application cannot be awoken by other participants. Unless the Background Notification service is used, as discussed in the next topic.

The Background Notification Service

Sometimes it is important to be awoken automatically when there is urgent data available that the application *must* respond to.

This can be done by connecting to a remote Background Notification Server, which then takes over the reading of important notifyable topics (by creating its own DataReaders), and signalling, i.e. waking up the sleeping application.

In practice, the application needs to connect to this Background Notification Server and communicate to this server with a proprietary suspend/resume protocol, which can be configured in 2 ways:

- Either by using a dedicated TCP connection to this server, or
- Piggy-backing the suspend/resume protocol on the existing TCP control channel which is used for controlling DDS/TCP/IP data connections.

Which domain is used and whether it is a separate or in-band TCP connection, is configured via DDS parameters, as will be explained later.

Client side API functions

A number of functions are available on the client side to control the wakeup procedure.

In order to let the server know which topics are important to be notified, following functions are available:

```
DDS_Activities_notify (DomainId id, const char *topic, const char *type);

DDS_Activities_unnotify (DomainId id, const char *topic, const char *type);
```

The first function registers a Reader Topic at the remote notification server, and the second function reverses this (for cleanup purposes, for example).

Note that these functions optionally take wildcards in their topic name and type specification, making it much simpler for the app developers to specify which topics/types are important. Simple Unix wildcards are permitted, so '*' and '?' characters can be used in name specifications.

The normal behaviour when a client is awoken is to do an automatic, i.e. implicit DDS_Activities_resume().

If this is not useful, i.e. sometimes the client application wants to be notified but is not really waking up yet, i.e. it just wants to send a message to the user, who will then decide whether to wake up or to continue sleeping.

For this reason, the following function is defined:

```
typedef void (*DDS_Activities_on_wakeup) (
 const char *topic_name,
 const char *type_name,
 unsigned char id [12]
);

typedef void (*DDS_Activities_on_connected) (
 int fd,
 int connected
);

DDS_EXPORT void DDS_Activities_register (
 DDS_Activities_on_wakeup wakeup_fct,
 DDS_Activities_on_connected connect_fct
);
```

If the function is installed properly, i.e. with valid callback functions, then the wakeup_fct will be called when the Notification Service has received data on one of the proxy readers, and the topic_name/type_name and id will be filled with the Topic Name, Type Name and Participant identity of the writer.

Note that this id is sufficient to identify the peer participant, since it is actually the GUIDPrefix, which is known as the DDS_BuiltinTopicKey_t key in the DDS_ParticipantBuiltinTopicData data structure.

It becomes the responsibility of the application developer to unfreeze the DDS component when this wakeup_fct is called. If not called eventually, the DDS software will no longer function properly.

The connect_fct will be called whenever the TCP connection to the Background Notification Service is established, or when it is closed. The idea is to tell the Operating System in the callback that this TCP connection is allowed to stay up when sleeping. Some Operating Systems (like iOS) require this, or the connection will be broken by the OS itself when it forces the application in a deep sleep.

Server side API functions

It is possible to be notified on the Background Notification Server itself that a client is connected or disconnected and it's state changes can be observed.

The following function allows this:

```
typedef enum {
 DDS_ACTIVITIES_CLIENT_ACTIVE,
 DDS_ACTIVITIES_CLIENT_SLEEPING,
 DDS_ACTIVITIES_CLIENT_DIED
} DDS_ActivitiesClientState;

typedef void (*DDS_Activities_on_client_change) (
 DDS_DomainId_t domain_id,
 DDS_BuiltinTopicKey_t *client_key,
 DDS_ActivitiesClientState state
);

DDS_EXPORT void DDS_Activities_client_info (
 DDS_Activities_on_client_change client_fct
);
```

The client_fct function, when installed is able to signal the following client states:

Table 2: Notification client states

State	Description
DDS_ACTIVITIES_CLIENT_ACTIVE	Either a new client is detected, or a transition from suspended to active occurred.
DDS_ACTIVITIES_CLIENT_SLEEPING	A connected client started sleeping, i.e. is in suspended state.
DDS_ACTIVITIES_CLIENT_DIED	A connected client became disconnected from the server. Either its battery is dead, or it is associated to a completely different network.

Environment variables for the Background Notification Service

As explained in the previous chapter, this Service requires some parameters in orde to be configured properly.

Following parameters can be distinguished:

Table 3:

Environment variable	Description
BGNS_DOMAIN	Specifies the DDS domain that is used for the Notification service. Both client and server must set this correctly or the service will used id 0.
BGNS_[SEC]PORT[6]	Used by the server to specify the TCP port number on which the server is listening for incoming connections. If set for the first time, a server for the notification service is started. Either a valid port number should be specified for a dedicated TCP server, or @ may be specified, which indicates a connection that is in-band, i.e. multiplexed on the connection of the DDS/TCP control channel server.
BGNS_[SEC]SERVER[6]	Used by the client to specify the TCP server URL of the remote Notification service. If set to @ the client connection will be in-band, i.e. multiplexed on the client connection of the DDS/TCP control channel. Once this variable is set, the client will attempt to connect to the remote server.

Note: The optional SEC prefix indicates that a secure TCP connection is needed for the server.

Note: The optional 6 suffix indicates that the TCP connection is over IPv6 instead of over IP.

Examples

export TDDS_BGNS_DOMAIN=7

Set the DDS domain to 7 for the Background Notification service.

export TDDS_BGNS_PORT=5500

Start a dedicated Background Notification server on port 5500.

export TDDS_BGNS_PORT=@

Start a Background Notification server multiplexed with the DDS TCP control channel.

export TDDS_BGNS_SERVER=www.technicolor.com/bgns:5500

Start a dedicated Background Notification service to the server at the given URL on port 5500.

export TDDS_BGNS_SERVER=@

Start a Background Notification client multiplexed with the DDS TCP control channel.