D-BOX DEVSIM OVERVIEW

A document describing content and concepts from D-BOX Live Motion SDK and DevSim

CONFIDENTIAL
226-989-0005-EN1_2

CONTENT

What is in this document?

D-BOX LiveMotion SDK is used to build gateways between simulation software applications as well as video game titles and D-BOX Motion Codes. This document is a description of D-BOX's Live Motion SDK for the DevSim application.

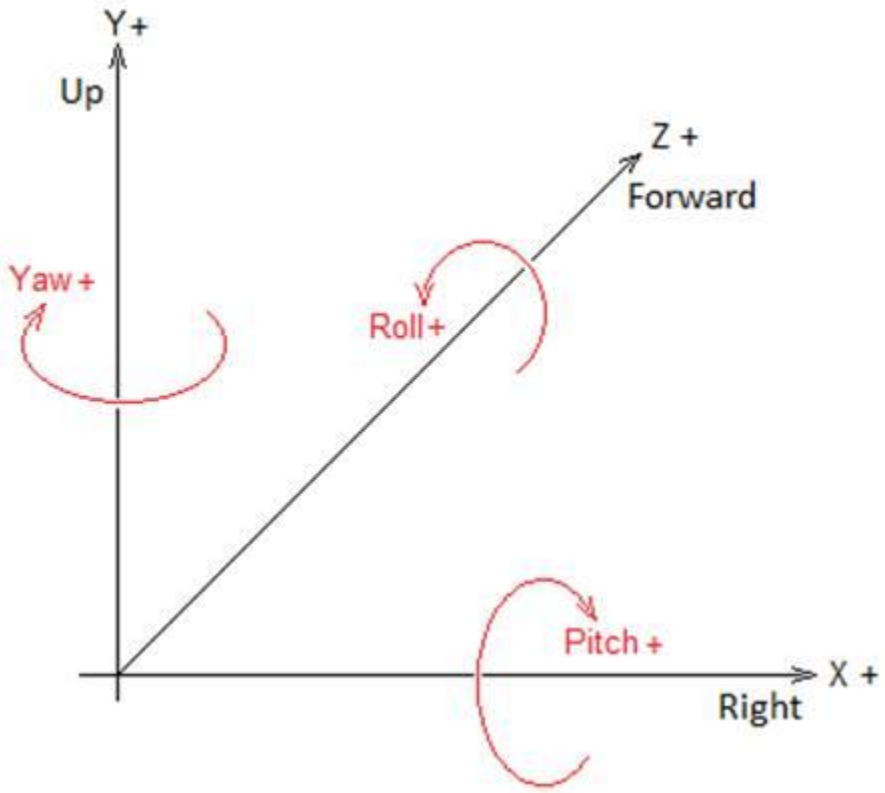
It contains a glossary of the ICD (interface control document) used to exchange data and events between a third party software application and D-BOX Motion Codes.

It also contains an appendix of a typical code sample to support a fast learning curve for integrators.

INITIAL REMARKS

Left handed Cartesian coordinate system

Note that the SDK uses a left-Handed Cartesian coordinate system as shown below:



Your APP_KEY

You need to use the APP_KEY “NameDevSim” (Take a look at the Sample.cpp to see where it is defined and all the other structures/calls). <Name> is a short prefix that refers to your company name.

METHODS

Structure layout registration macros

Use these macros to define the fields layout, type and meaning in a struct. They will create a static GetStructInfo() method in your struct suitable for registration with the RegisterEvent method. Later on, an instance of the registered struct can be supplied to the PostEvent method.

Sample usage: (note that there is no ";" at end of macro calls)

```
struct SimConfig {
        dbox::F32 MasterGain;
        dbox::F32 MasterSpectrum;

        DBOX_STRUCTINFO_BEGIN()
        DBOX_STRUCTINFO_FIELD(SimConfig, MasterGain, dbox::FT_FLOAT32, dbox::FM_MASTER_GAIN_DB)
        DBOX_STRUCTINFO_FIELD(SimConfig, MasterSpectrum, dbox::FT_FLOAT32, dbox::FM_MASTER_SPECTRUM_DB)
        DBOX_STRUCTINFO_END()
};
```

Then SampleConfig::GetStructInfo() can be supplied to RegisterEvent method.

WARNING: Your StructInfo layout must match exactly the layout of your struct. These macros facilitate this because they can both be maintained close to each other in your source code.

Here is more information about the structs in Sample.cpp:

- SimConfig: (this struct is a dbox EM_CONFIG_UPDATE type)
- MotionConfig: (this struct is a dbox EM_CONFIG_UPDATE type)
- MotionData: (this struct is a dbox EM_FRAME_UPDATE type)

SDK methods

dbox::LiveMotion::Initialize(PCCHAR sAppKey, U32 nAppBuildVersion);

- Global initialization. It must be called once at start-up. Initialize/Terminate calls must be balanced.
- Parameters:
  - sAppKey: should be set to a constant char * that uniquely identifies your application (usually assigned by D-BOX).
  - nAppBuildVersion: should be set to an integer representing your build number.

dbox::LiveMotion::Terminate();

- Global termination. It must be called at the end and it is very important to do so.
- Initialize/Terminate calls must be balanced.

dbox::LiveMotion::Open();

- Opens the motion output device. This will normally prepare the motion platform for output. Open/Close calls must be balanced.

dbox::LiveMotion::Close();

- Closes the motion output device. This will normally release the motion platform and immediately stop any pending output.
- Open/Close calls must be balanced.

dbox::LiveMotion::Start()

- Gently starts motion activity based on PostEvent calls and internal state of Motion Code logic. A fade-in will be applied for smooth transition. It should be called as soon as the end-user can interact with the virtual environment, like start of simulation/level, resume from pause, etc.

dbox::LiveMotion::Stop()

- Gently stops motion activity. A fade-out will be applied for smooth transition. It should be called as soon as the end-user can no longer interact with the virtual environment, like end of simulation/level, pause, etc.

dbox::LiveMotion::ResetState()

- Resets the internal state of the Motion Code logic. It should normally be called before Start or after Stop (except for pauses) when the environment or context changed.

dbox::LiveMotion::RegisterEvent(U32 nPermanentEventKey, EEventMeaning eMeaning, const StructInfo & oGetStructInfo)

- Registers an event or message with related data or attributes. Events are used to communicate configuration or real-time data that can be used for motion generation. You can combine many data fields in a structure and post them at once in a single event. To do so you must create a structure and use the DBOX_STRUCTINFO_* macros to create structure layout information that will be used by this RegisterEvent method.
- This way, the D-BOX Motion Code logic will be able to interpret your structure data on subsequent PostEvent calls. To post these events you will need to use the 2-parameter PostEvent method.
- Parameters:
  - nPermanentEventKey: This is your unique reference for later posting this event. Once a Permanent Event Key is used in motion generation, it must not change with future releases of your application to prevent compatibility breaks.
  - eMeaning: This is the general meaning of your event.
  - oGetStructInfo: For a structure with appropriate DBOX_STRUCTINFO_* layout, use [your struct name]::GetStructInfo().

dbox::LiveMotion::PostEvent(U32 nPermanentEventKey, Struct oStruct)

- Sends an event from your application to all sessions.

DATA FIELDS

The following is the list of known meanings that can be associated to fields. This association will be used during event registration.

Field types

- FRAME FIELDS

  Frame Fields should be posted at every simulation frame or game loop frame.

  Their state is global and will be maintained by the Motion Code between frames. Even if a same FRAME FIELD is registered in different event structures, it will update the same global state in the Motion Code.

- CONFIGURATION FIELDS

  Configuration fields can be posted when required or when a change occurs.

  Their state is global and will be maintained by the Motion Code between updates. Even if a same CONFIGURATION FIELD is registered in different event structures, it will update the same global state in the Motion Code.

Global Mixer Fields

- FM_MASTER_GAIN_DB : (configuration field) Master gain applied at the final output, in decibel. It is recommended to expose this setting in the application's user interface. (0dB is the default level).
- FM_MASTER_SPECTRUM_DB : (configuration field) Motion/Vibration balance control, in decibel. It is recommended to expose this setting in the application's user interface. Value must be between -20dB and +20dB: -20dB being maximum vibration (high frequencies) attenuation, +20dB being maximum motion (low frequencies) attenuation and 0dB being normal signal (full spectrum).

General Vehicle Engine fields

- FM_ENGINE_RPM_MAX : (configuration field) Absolute maximum RPM of vehicle engine.
- FM_ENGINE_RPM_IDLE : (configuration field) Common idle RPM of vehicle engine.

- FM_ENGINE_RPM : (frame field) Instantaneous engine rotation speed in RPM.
- FM_ENGINE_TORQUE_MAX : (configuration field) Maximum engine torque in Newton meters (1 foot-pound = 1.35581795 newton-meter).
- FM_ENGINE_TORQUE: (frame field) Instantaneous engine torque in Newton meters (1 foot-pound = 1.35581795 newton-meter). Torque should be negative when braking on engine compression.

Raw Motion Data fields

Saturation is a common issue. It is when the total value of one actuator goes beyond 1 or -1. Meaning that you cannot have +1 or -1 on Roll, Pitch and Heave. You must make sure the total value for an actuator is equal or lower than +1, or higher for -1.

- FM_RAW_ROLL : (frame field) Left roll value transmitted directly to the Motion Output. Value should be between -1 and +1: -1 being max leaning to the right and +1 being max leaning to the left.
- FM_RAW_PITCH : (frame field) Forward pitch value transmitted directly to the Motion Output. Value should be between -1 and +1: -1 being max backward pitch and +1 being max forward pitch.
- FM_RAW_HEAVE : (frame field) Up heave value transmitted directly to the Motion Output. Value should be between -1 and +1: -1 being bottommost position and +1 being topmost position.

APPENDIX

Sample delivered with D-BOX DevSim

```
#include <cstdio>
#include <conio.h>
#include <Windows.h>
#include "LiveMotionSdk/dboxLiveMotion.h"
#include <math.h> // For sin

#pragma comment(lib, DBOX_LIVEMOTION_LIB)

const char* const APP_KEY = "DevSim";
const dbox::U32 APP_BUILD = 1000;
const double PI = 3.141592654f;

struct SimConfig {
	dbox::F32 MasterGain;
	dbox::F32 MasterSpectrum;

	DBOX_STRUCTINFO_BEGIN()
	DBOX_STRUCTINFO_FIELD(SimConfig, MasterGain, dbox::FT_FLOAT32, dbox::FM_MASTER_GAIN_DB)
	DBOX_STRUCTINFO_FIELD(SimConfig, MasterSpectrum, dbox::FT_FLOAT32, dbox::FM_MASTER_SPECTRUM_DB)
	DBOX_STRUCTINFO_END()
```

```
};

struct MotionConfig {
	dbox::F32 EngineRpmIdle;
	dbox::F32 EngineRpmMax;
	dbox::F32 EngineTorqueMax;

	DBOX_STRUCTINFO_BEGIN()
	DBOX_STRUCTINFO_FIELD(MotionConfig, EngineRpmIdle, dbox::FT_FLOAT32, dbox::FM_ENGINE_RPM_IDLE)
	DBOX_STRUCTINFO_FIELD(MotionConfig, EngineRpmMax, dbox::FT_FLOAT32, dbox::FM_ENGINE_RPM_MAX)
	DBOX_STRUCTINFO_FIELD(MotionConfig, EngineTorqueMax, dbox::FT_FLOAT32, dbox::FM_ENGINE_TORQUE_MAX)
	DBOX_STRUCTINFO_END()
};

struct MotionData {
	dbox::F32 Roll;  // -1.0 to 1.0
	dbox::F32 Pitch;  // -1.0 to 1.0
	dbox::F32 Heave;  // -1.0 to 1.0
	dbox::F32 EngineRpm; // Depends on MotionConfig EngineRpmIdle and EngineRpmMax
	dbox::F32 EngineTorque; // Depends on MotionConfig EngineTorqueMax

	DBOX_STRUCTINFO_BEGIN()
	DBOX_STRUCTINFO_FIELD(MotionData, Roll, dbox::FT_FLOAT32, dbox::FM_RAW_ROLL)
	DBOX_STRUCTINFO_FIELD(MotionData, Pitch, dbox::FT_FLOAT32, dbox::FM_RAW_PITCH)
	DBOX_STRUCTINFO_FIELD(MotionData, Heave, dbox::FT_FLOAT32, dbox::FM_RAW_HEAVE)
	DBOX_STRUCTINFO_FIELD(MotionData, EngineRpm, dbox::FT_FLOAT32, dbox::FM_ENGINE_RPM)
	DBOX_STRUCTINFO_FIELD(MotionData, EngineTorque, dbox::FT_FLOAT32, dbox::FM_ENGINE_TORQUE)
	DBOX_STRUCTINFO_END()
};


/// These are your unique event ids that you'll use when calling PostEvent.
enum AppEvents {
	SIM_CONFIG = 1000,
	MOTION_CONFIG = 2000,
	MOTION_DATA = 3000,
};

/// This function is used to prevent possible error when the application ends before the normal execution.
/// This includes Closing the console window, CTRL-C, Windows Shutdown, Log Off...
bool WINAPI OnAbnormalTerminate(DWORD /*dwCtrlType*/) {
	dbox::LiveMotion::Terminate();
	return true;
}

int main(int , char* []) {
	SetConsoleCtrlHandler((PHANDLER_ROUTINE)OnAbnormalTerminate, true);

	// Create a sample sinus signal
	float adSinusSignal[1000]; // 1000 samples
	for (int nIndex = 0; nIndex < 1000; nIndex++) {
		adSinusSignal[nIndex] =  static_cast<float>(sin(2.0f*PI*nIndex/1000.0f));
	}

	// Initialization and registration should be done only once.
	dbox::LiveMotion::Initialize(APP_KEY, APP_BUILD);
	dbox::LiveMotion::RegisterEvent(SIM_CONFIG, dbox::EM_CONFIG_UPDATE, SimConfig::GetStructInfo());
	dbox::LiveMotion::RegisterEvent(MOTION_CONFIG, dbox::EM_CONFIG_UPDATE, MotionConfig::GetStructInfo());
	dbox::LiveMotion::RegisterEvent(MOTION_DATA, dbox::EM_FRAME_UPDATE, MotionData::GetStructInfo());

	// Registration completed, open motion output device.
	// Open can be called once, after end of registration.
	dbox::LiveMotion::Open();
	{
```

```
// Motion System initialization
Sleep(10000);

// Sim Start
// It is always good to ResetState before each session.
dbox::LiveMotion::ResetState();

SimConfig oSimConfig;
oSimConfig.MasterGain = 0; // 0dB
oSimConfig.MasterSpectrum = 0; // 0dB
dbox::LiveMotion::PostEvent(SIM_CONFIG, oSimConfig);

MotionConfig oMotionConfig;
oMotionConfig.EngineRpmIdle = 750.4f;
oMotionConfig.EngineRpmMax = 3420.2f;
oMotionConfig.EngineTorqueMax = 447.42f;
dbox::LiveMotion::PostEvent(MOTION_CONFIG, oMotionConfig);

MotionData oMotionData;
oMotionData.Roll = -0.3f;
oMotionData.Pitch = 0.2f;
oMotionData.Heave = 0.3f;
oMotionData.EngineRpm = 1000;
oMotionData.EngineTorque = 175;
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

// Start of sim, this will fade-in actual motion.
dbox::LiveMotion::Start();

oMotionData.Roll = 0;
oMotionData.Pitch = 0;
oMotionData.Heave = 0;
oMotionData.EngineRpm = 0; // Mute RPM
oMotionData.EngineTorque = 0;
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

Sleep(5000);

// Simulate Sinus signal
for (int nSinLoop = 0; nSinLoop < 5; nSinLoop++) { // Play sinus 5 times
        for (int nIndex = 0; nIndex < 1000; nIndex++) {
                oMotionData.Heave = adSinusSignal[nIndex]; // Simulate Heave
                //oMotionData.Roll = adSinusSignal[nIndex]; // Simulate Roll
                //oMotionData.Pitch = adSinusSignal[nIndex]; // Simulate Pitch
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
                Sleep(5);
        }
}

// Continue posting Motion data...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
// ...
dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
```

```
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

                Sleep(5000);

                oMotionData.Roll = 0.3f;
                oMotionData.Pitch = -0.2f;
                oMotionData.Heave = -0.3f;
                oMotionData.EngineRpm = 2000;
                oMotionData.EngineTorque = 195;
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

                Sleep(5000);

                // Reset State
                oMotionData.Roll = 0;
                oMotionData.Pitch = 0;
                oMotionData.Heave = 0;
                oMotionData.EngineRpm = 0;
                oMotionData.EngineTorque = 0;
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

                Sleep(5000);

                // Pause, this will fade-out motion.
                dbox::LiveMotion::Stop();

                // Resume from pause, this will fade-in actual motion.
                dbox::LiveMotion::Start();
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);
                // ...
                dbox::LiveMotion::PostEvent(MOTION_DATA, oMotionData);

                // End of level, this will fade-out motion.
                dbox::LiveMotion::Stop();

                // Level End
        }
        // Close motion output device.
        dbox::LiveMotion::Close();
        // Terminate
        dbox::LiveMotion::Terminate();

        printf("\nEnded, press any key...\n");
        _getch();

        return 0;
}
```