Documentation

GNA Updater SDK v1.0.0 for Windows

Documentation

GNA Updater SDK for Windows

VERSION: 1.0.0

Release date: 03-11-2022

Scope

This release note applies to GNA Updater SDK software version 1.0.0 for Windows operating system.

Software description

The Windows GNA Updater SDK is designed to provide third party Microsoft Windows application developers the ability to interface with BlueParrott Bluetooth headsets via a USB dongle.

Jabra GN

Documentation

Contents

Windows (OS) supported

Windows Operating Systems (OS)
Windows 8.1
Windows 10
Windows 11

System prerequisites

Microsoft .NET 4.5 Full Framework
Microsoft .NET 4.7.2 Developer Pack
Microsoft Visual C++ 2005 Redistributable Package (x86)
Microsoft Visual C++ 2012 Redistributable Package (x86)
Microsoft Visual C++ 2013 Redistributable Package (x86)
Microsoft Visual C++ 2015 Redistributable Package (x86)
Microsoft Visual C++ 2017 Redistributable Package (x86)

Device models supported

Model	PID	Chip	DFU Architecture
M300-XT	0x29	5126 based	USB HID DFU
C300-XT		8670	USB QC DFU

B450-XT II	0x25	8670	USB QC DFU
C350-XT		8670	USB QC DFU

Limitations

Only 32-bit version of SDK is available at the moment, meaning that only 32-bit applications would be able to integrate the SDK. Developers can still use “AnyCPU” Visual Studio build configurations for future possibility to switch to 64-bit, but should have “Prefer32Bit” flag set for the build.

GNA Updater SDK

The Updater SDK package includes the following development tools:

- SDK Updater Library (GNAUpdaterSDK.dll and other dependency libraries)
- Notes and documentation.

How the SDK Works

The GNAUpdaterSDK.dll provides the ability to update BlueParrott products that have a USB connection in Windows OS. The main targets for upgrades at this time are CSR 8670 and 5126 based products.

Including the SDK in the project

In your solution add a reference for the GNAUpdaterSDK.dll. This will allow you to use the whole functionality of GNA Updater SDK library.

How to add GNAUpdaterSDK.dll to the project:

1) Copy GNAUpdaterSDK.dll and other dependent libraries to the required folder.
2) Go to references in GNAUpdaterSDK_Demo project
3) Right click on the references and select Add References

4) Select browse tab and click browse
5) Select GNAUpdaterSDK.dll location and click OK (figure 1)
6) Click OK

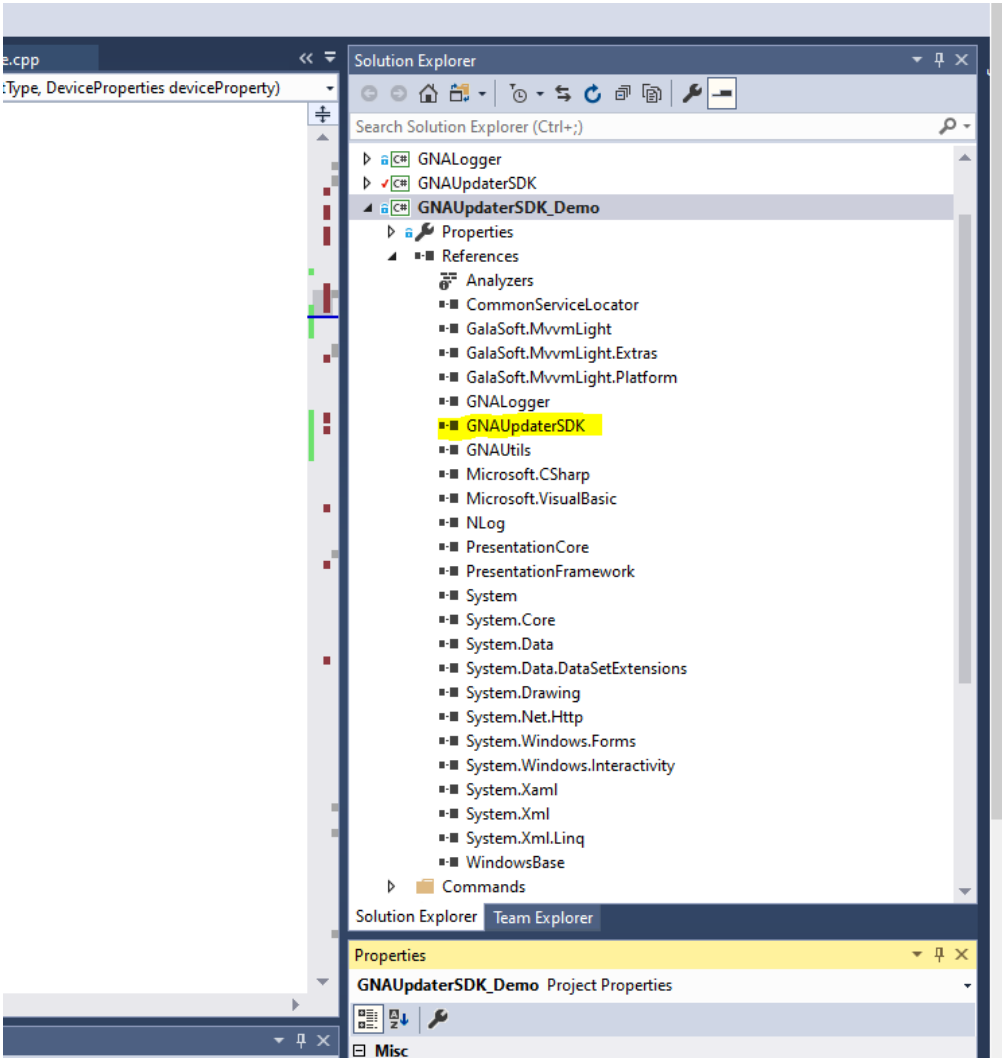

Figure 1. GNAUpdaterSDK.dll

SDK Configurable settings

GNA Updater SDK has a set of configurable properties that can be overridden by the end user. These settings are affecting different aspects of SDK logic described in the table below:

Property Name	Type	Default value	Description
OfflineModeEnabled	Boolean	False	Property value defines whether Offline Mode is enabled. If set to True, no connection to the internet will be done and only local data is used.
BlueParrottVendorIDs	String	10978	Property value contains comma separated values that are used for filtering headsets connected to the system.
BlueParrottProductNames	String	41:M300-XT,22:C-300-XT,37:B450-XT II	Property value contains comma separated values that are used for setting headsets friendly name in Opaque Mode (when there is no internet connection and local database).
DatabaseCheckIntervalMS	Integer	300000	Property value determines how often data is being checked and updated

			from the server. Value should be an integer defining an interval in milliseconds.

Config file should have a name “GNAUpdaterSdk.cfg” and should be in the same directory as the main executable.

Config file can contain commented lines, starting with the hash symbol (#). Commented lines are ignored.

Config file should contain lines in the following format: <Property Name>=<Property value>

Note: spaces or tabs can be added to separate name and value (see Figure 2 example)

If configuration file is not available default values are used.

If configuration file has an invalid data default values are used, and error is logged.

Below is the sample config file:

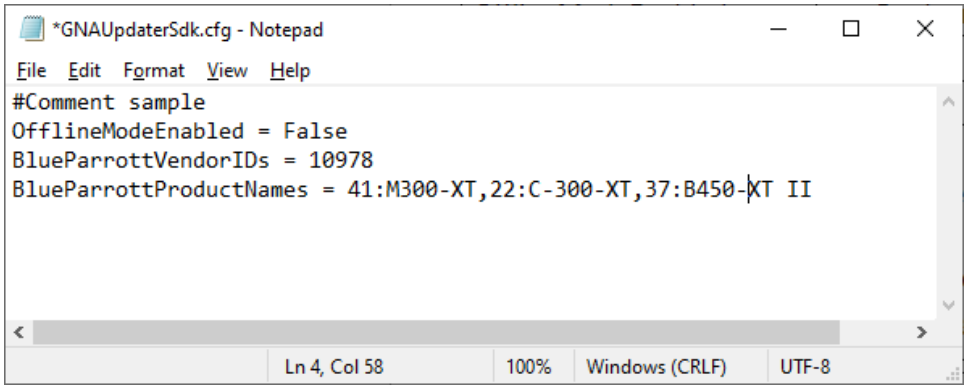

Figure 2. Sample config file

Logging path

The SDK stores GNAUpdaterSDK.log in the logs folder. This folder is next to the SDK library. If there are any issues with SDK, these logs will help the development team to reproduce and analyze the issue.

SDK API

The SDK is mainly implemented as a Windows® DLL. The SDKs main function is to execute commands from its client App. But it also maintains some state and does some background processing and makes some async call-backs to the client App. It starts operations when the client App commands it to. See Figure 3.

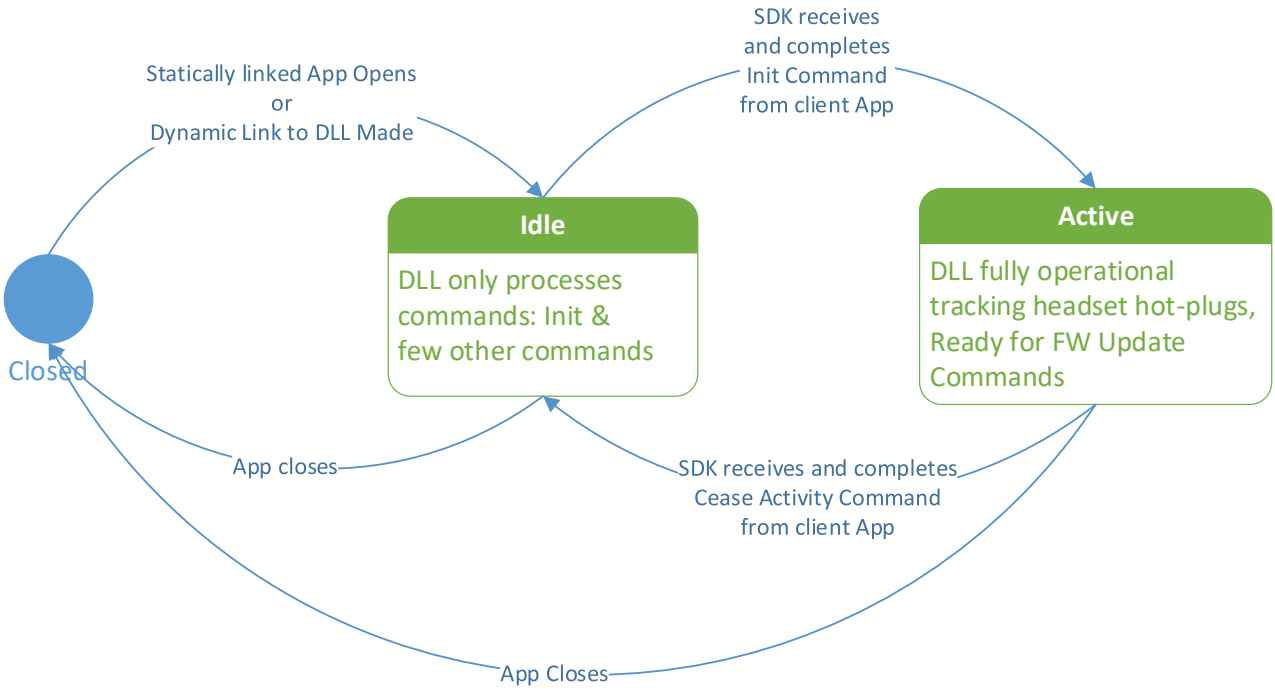

Figure 3. SDK states

Idle State

When the SDK starts it is in the Idle state. In the Idle state it is only waiting for commands from the client App. The only commands it can successfully process are: Get SDK Version (sync and async) and Initialize.

```
public static int Initialize(InitializeCallback callback, ulong context, SdkDataSourceMode dataSourceMode, int logLevel) - After the SDK completes an Initialize command it is in the Initialized state. In the Initialized state it can process all commands except the Initialize command

where:

callback – provides an instance of API, context and result code

context - The context parameter is a value passed in an Async Call-in Function. The DLL passes that same value out for the corresponding Async Call-back Function.

dataSourceMode – an option that decides whether SDK uses online data provider or uses only local offline DB. If ConfigFile (0) option is selected, this setting will be read from config file (see “SDK Configurable settings” section). If Online (1) option is selected, an
```

online data source will be used and internet connection is necessary. In Offline (2) mode, the snapshot of the local database will not be updated, and access to the Internet is not required for the SDK to work. Default value is ConfigFile (0).

logLevel – an option that defines the minimum severity of the record to be logged. Below is the list of available log levels. Default value is Info (3).

```
Fatal = 0,
Error = 1,
Warn = 2,
Info = 3,
Debug = 4,
Trace = 5
```

public static int **GetSDKVersion**(out string version) – provides the SDK Version string

where:

version – provides the SDK version

public static int **GetSDKVersionAsync**(GetSDKVersionCallback callback, ulong context) – provides asynchronously the SDK Version string.

where:

version - provides the SDK version

context - The context parameter is a value passed in an Async Call-in Function. The DLL passes that same value out for the corresponding Async Call-back Function

Active State

After the SDK completes an Initialize command it is in the Active state. In the Active state it can process all commands except the Initialize command. It maintains the following information:

1) int **GetAvailableDevices**(out Device[] devices) – provides a list of connected headsets

where:
devices – array of connected devices

2) int **GetSerializedAvailableDevices**(out string serializedDevices) – provides a list of connected headsets

where:
serializedDevices – serialized array of connected devices

3) int **GetHeadsetDetails**(int DeviceID, out string deviceDetails) - provides headset details like: Model name, model number, FW version, VID, PID, unique identifier

where:
DeviceID – headset unique identifier
deviceDetails – serialized device details

4) int **GetDeviceName**(int DeviceID, out string deviceName) - provides headset name

where:
DeviceID – device unique identifier
deviceName – headset name

5) int **GetCurrentFirmwareVersion**(int DeviceID, out string deviceFWVersion) – provides current headset firmware version

where:
DeviceID – device unique identifier
deviceFWVersion – current headset firmware version

6) int **GetLatestFirmwareVersion**(int DeviceID, out string deviceFWVersion) – provides latest headset firmware version

where:
DeviceID – device unique identifier
deviceFWVersion – latest headset firmware version

7) int **GetAvailableFirmwareVersions**(int DeviceID, out string[] deviceFWVersions) – provides a list of available headset firmware version

where:
DeviceID – device unique identifier
deviceFWVersions – latest headset firmware version

1) int **GetHeadsetImage**(int DeviceID, out byte[] deviceImage) – provides current headset image

where:
DeviceID – device unique identifier
deviceImage –headset image

2) int **InstallDrivers**()- Install necessary drivers
3) int **StartUpgrade**(int DeviceID, UpdateStatusCallback statusCallBack, UpdateDeviceCallback callback, ulong context, string firmwareFilePath = null, bool isReportOnly = false) - Client App commands that the SDK update FW for specified headsets. The command specifies either a local file path or a FW version from the

FW Image Repo Server. The SDK would either load the local file FW image or download the FW image from the server to complete this command.

where:
DeviceID – device unique identifier
statusCallBack – publishes device upgrade status (including update state, progress of each stage
callback – publishes device upgrade status (including feasibility of update, error codes, etc)
context – The context parameter is a value passed in an Async Call-in Function. The DLL passes that same value out for the corresponding Async Call-back Function.
firmwareFilePath – location of the local firmware image, default value is false. When set to false, SDK will try to find and download the latest firmware from online server.
isReportOnly – decides whether actual upgrade happens. If set to true, no upgrade happens, in this mode function can be used to check if device is legible for an upgrade.

4) int **CancelFWUpgrade**(int DeviceID) – cancel update for the whole headset group

where:
DeviceID – device unique identifier

5) int **CeaseActivity**() - The client commands the SDK to return to the Idle state by issuing the Cease Activity Command. The Cease Activity Command does the opposite of the Initialize command.
6) int **SubscribeEvents**(AttachedDeviceCallback Attached, DetachedDeviceCallback Detached, DevicePropertiesUpdatedCallback DevicePropertiesUpdated, DBUpdatedCallback DBUpdated = null, DefferedCallback Deffered = null, NoOpCallback Noop = null) – subscribe for necessary events

Where:
AttachedDeviceCallback – published when a new headset is connected
DevicePropertiesUpdatedCallback – published when the headset is disconnected
DBUpdatedCallback- published when the database is updated
DefferedCallback – published when the deferred error occurs
NoOpCallback- is required for testing purposes

7) int **ClearSubscriptions**() - clearing all subscriptions
8) int **SendNoOp**(string text) – Client App commands that the SDK send a No-op event. (This is for testing and development.)

where:
Text – custom text, which will be returned in the callback

14) int **CancelAsyncCallback**() – cancels asynchronous callback
15) int **TestServerConnectionAsync**(TestServerAccessCallback callback, ulong context) - Client App commands that the SDK attempt to access the server and respond about success or failure

where:
callback – publishes server connections status
context - The context parameter is a value passed in an Async Call-in Function. The DLL passes that same value out for the corresponding Async Call-back Function.

16) int **RefreshConfig**(RefreshConfigCallback callback, ulong context) – command forces a DB update from online server. In case of Offline Mode, ForbiddenInOfflineMode error will be returned.

where:

```
        callback – publishes when configuration was updated or failure happened.
        context - The context parameter is a value passed in an Async Call-in Function.  The DLL
passes that same value out for the corresponding Async Call-back Function.
```

Background Processing

The only processing the SDK does autonomously is it subscribes to or monitors Windows device arrival and device removal notifications and processes them. It effectively detects USB headset hot-plugs and keeps its roster of connected headsets current. It may query the server for Headset Model Information. Everything else the SDK does is directly part of processing a specific command from the client App.

SDK error codes

Below is the list of error codes returned by SDK with a brief description of the error.

Error code	Friendly name	Description
0	Success	Operation success.
-1	UnknownError	Unexpected error. Please report an error to the SDK development team.
-2	NotInitialized	Use of this operation is deprecated with non-initialized SDK. Make sure you call SDK Initialize method before using this API.
-3	AsyncInvocationFailed	Error happened during async operation invocation. Please retry calling the operation. If the problem persists, please report an error to the SDK development team.
-4	UserInterrupted	Operation was interrupted(cancelled) by user.
-5	ForbiddenInOfflineMode	Operation is forbidden in offline mode. Disable offline mode to use this operation.

-10	DLLDependencyMissing	Necessary dll dependency file is missing. Please check that SDK is deployed correctly.
-11	OpenTestEngineFailed	Dependency module error. Unable to get device configuration data.
-12	ConnectedDevicesDetectionFailed	Device detection failed. If the problem persists, please report an error to the SDK development team.
-13	CertificateInstallationFailed	Certificate installation failed. If the problem persists, please report an error to the SDK development team.
-14	ServerConnectionFailed	Cannot reach data web server. Please check the internet connection.
-15	ServerOperationFailed	Data server request failed. Please retry the operation, if the problem persists, please report an error to the SDK development team.
-16	ReadDBFailed	Read offline data failed. The likely reason is that offline data got corrupted. If the problem persists, try clearing offline data cache (“offline” directory and all its contents). If no online data is available or if operating in Offline mode, the application will fall back into Opaque Mode.
-20	LoadDevicesFailed	Unable to load device list. If the problem persists, please report an error to the SDK development team.
-21	AccessViolation	Memory operation failed within the dependency module. If the problem persists, please report an error to the SDK development team.
-22	ReadPSKeyFailed	Failed to read device configuration data. The likely reason is corrupted device storage data.
-23	WritePSKeyFailed	Failed to write device configuration data. If the problem persists, please report an error to the SDK development team.

-24	ReadHIDDataFailed	Failed to read HID data from the device. Please check device connection and firmware integrity.
-25	GetDeviceNameFailed	Failed to get device name via PSKey read operation.
-26	GetFirmwareVersionFailed	Failed to get device Firmware version information.
-27	GetDeviceHIDInterfaceNumbersFailed	Failed to get device interface numbers while trying to read HID data.
-28	GetDeviceUSBInfoFailed	Failed to get device USB information.
-29	GetDeviceArrivalProcessingFailed	Failure during device arrival handling. If the problem persists, please report this error to the SDK development team.
-30	GetDeviceRemoveProcessingFailed	Failure during device removal handling. Please check device connectivity and firmware. If the problem persists, please report this error to the SDK development team.
-31	GetDevicePathFailed	Failed to get device path while handling device arrival/removal events. Please check device connection and firmware integrity.
-101	UnknownError	Unexpected error in Device Service. Please report an error to the SDK development team.
-102	DatabaseUnavailable	Database is unavailable. This error could happen if the DB was not yet initialized before the API call. Make sure to subscribe to DBUpdated event to avoid this situation. Also, this can happen if DB file was corrupted, and SDK is unable to reach web server to get online data (or if in Offline mode). If the problem persists, try clearing offline data cache (“offline” directory and all its contents).
-103	UnknownOperatingSystem	Unknown/unsupported Operating System was detected. Please check that you are using a supported

		OS, see list of supported systems in “Windows (OS) supported”
-104	DataServerError	Failed to read data from online data server, please check the logs for data server related errors.
-105	InvalidData	Invalid data encountered while performing the API operation. If the problem persists, try clearing offline data cache (“offline” directory and all its contents).
-106	SwitchToDFUModeFailed	Failed to switch device to classic DFU mode to perform a classic DFU upgrade. Please check the device firmware integrity.
-107	SerializationFailed	An error happened during object serialization. Please report this issue to the SDK development team.
-108	DeviceNotFound	SDK was unable to find a connected device by given device ID. Please check that you use correct data in API calls. If the problem persists, please report this error to the SDK development team.
-109	DeviceModelNotFound	SDK was unable to read device model from the device. Please check the device firmware integrity.
-110	DeviceReconfiguringFailed	Device upgrade failed during reconfiguration phase. Please check logs for more information.
-111	DeviceDownloadingFailed	Device upgrade failed during downloading phase. Please check logs for more information.
-112	DeviceVerifyingFailed	Device upgrade failed during verifying phase. Please check logs for more information.
-113	CoreConfigurationNotFound	SDK was not able to find matching core configuration for device model. If the problem persists, try clearing offline data cache (“offline” directory and all its contents).
-114	DriverNotFound	No driver package file was found in the cache and SDK is unable to retrieve driver data from online data server.

-115	DriverFileCorrupted	Driver package file was found, but SDK failed to read data from the file. If the problem persists, try clearing offline driver cache (“driver” directory and all its contents).
-116	DriverInstallerMissing	Driver installer was not found in driver package. If the problem persists, try clearing offline driver cache (“driver” directory and all its contents).
-117	DriverInstallerError	Driver installation failed due to an internal installer error. Please check the logs for more information.
-118	DriverInstallationFailed	Driver installation failed. Please check the logs for more information.
-119	FirmwareNotFound	Firmware data was not found in the database. If the problem persists, try clearing offline data cache (“offline” directory and all its contents).
-120	FirmwareFileCorrupted	Firmware data is corrupted, either invalid file format or firmware hash does not match the actual data. If the problem persists, try clearing offline data cache (“offline” directory and all its contents).
-121	FirmwareFileMissing	Firmware file was either not found at given location or file failed to download from online data server.
-122	FirmwareNotValidForMode	Firmware is not valid for device upgrade mode. Please make sure the correct firmware file was set in API call. If using firmware from online data server, please report this error the the SDK development team.
-123	UpgradeInProgress	Unable to start a new upgrade due to already running upgrade operation.
-124	NoActiveUpgrade	SDK is unable to cancel firmware upgrade operation. No active upgrade was found.
-125	DeviceUpgradeNotSupported	Upgrade is not supported for the selected device.

-126	NotCompatibleDevicesForUpgrade	Invalid devices were detected and selected for an upgrade. Please report this error to the SDK development team.
-127	UpgradeAborted	Upgrade was cancelled by the user.
-201	UnknownError	Unexpected error in Data Server Service. Please report an error to the SDK development team.
-202	CertificateNotInstalled	Server certificate is not installed, so SDK is not able to interact with online data server. Please check the logs for more information.
-203	InvalidServerResponse	Online data server returned an error. Please check the logs for more information.
-204	ServerNotAvailable	Online data server is not available. Please check that client is connected to the internet and the server connection is not being blocked by the firewall or anti-malware software.

Demo Application

There is a sample app of simple integration available for Windows. Allowing the developer to get up and running quickly while using the GNA Updater SDK.

This simple demo includes the following features:

- Display connected headsets;
- Display information about connected headsets;
- Display headset firmware version;
- Test server confection;
- Subscribe events;
- Clear subscriptions;
- Offline mode;
- Send No-op events;
- Get current headset name;
- Get current headset firmware version;
- Get latest headset firmware version;
- Get available headset firmware versions;
- Get SDK version sync and async;
- Install drivers;
- Initialize/Cease activity;
- Upgrade headset from the server and local file;

Main Window

The main windows contain the following information (figure 4):

1) SDK version – current SDK version (sync version)
2) Demo version – current demo version
3) Initialize/Cease Activity – Initialize or Cease Activity – the appropriate API methods are called
4) Install Drivers – installing the necessary drivers on the system
5) Browse – provides the ability to select the firmware image for the current device group.
   The local image has a higher priority compared to the server image.
6) The device list – provides a device list with available headsets. Each device contains the following info: Name, Vendor Id, Product ID, USB Path
7) Two buttons: Info, Update, Cancel Update
   Update – provides an update of the selected device group (with the same product id)
   Cancel Update – provides canceling the firmware update (of the whole headset group)

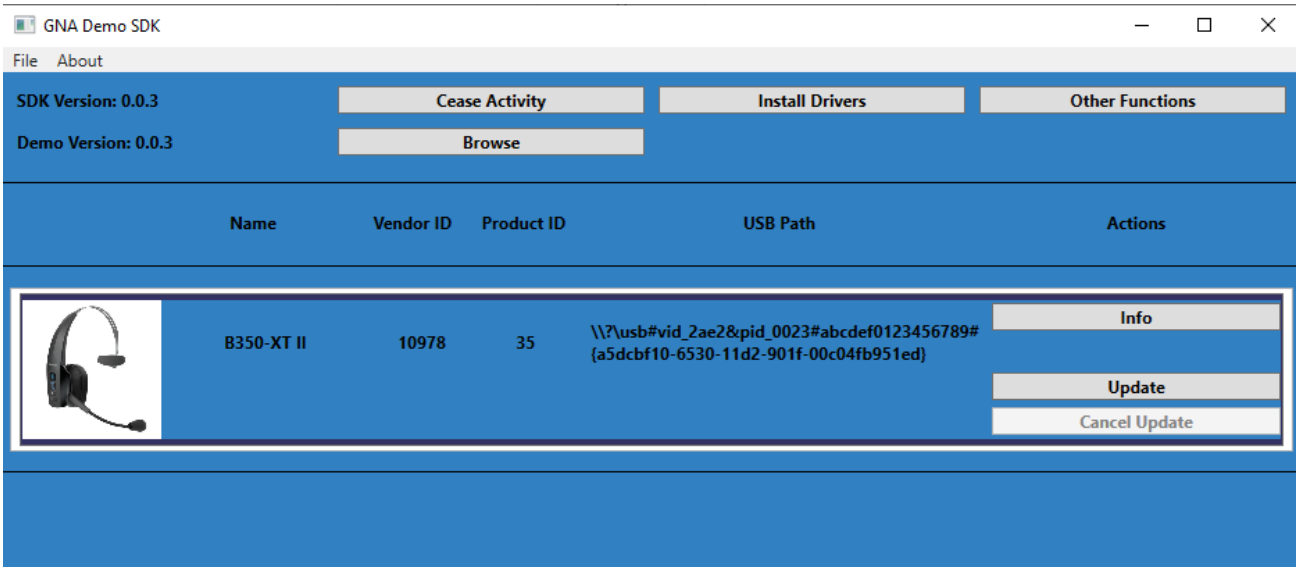

Figure 4. Main Window

Info Button

The Info Button contains the general information about the current headset (figure 5):

1) Device ID
2) Name
3) Firmware version
4) Product ID
5) Vendor ID
6) Device Location
7) Device Path
8) Device Class
9) Container ID

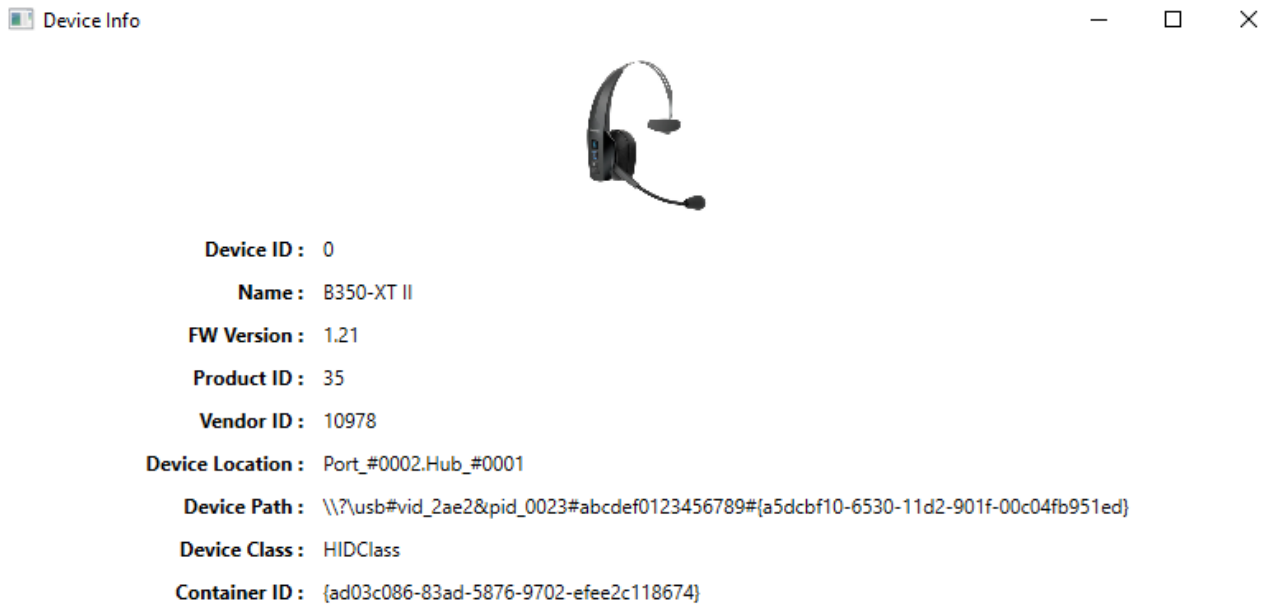

Figure 5. Info button

Other Functions

Other functions provide the calling of other SDK methods (figure 6):

1) Test server connection: (Ok or Error codes). (Appropriate API method).
2) Subscribe events - Subscribe for all events (Appropriate API method).
3) Clear Subscriptions - Cancel update for the whole headset group (Appropriate API method).
4) Offline Mode - The SDK is designed to be able to operate in offline mode without accessing the server. If offline mode is enabled, the SDK will not attempt to connect to the server.
5) Send No-op - The client app requests this event (usually for testing and development purposes) via a Send No-op Event command. After the SDK completes the Send No-op Event command it sends the No-op event.
6) Cancel Async Call-back - The client app has issued an Async Command that has not been completed yet. The client app commands to cancel that command and receive no call-back for it.
7) Get Current Device Name – The client app queries about device friendly configured on the server side.
8) Get Current Device Firmware Version – The client app queries about which FW versions for a specific headset are available on the server.
9) Get Interval Current Device Firmware Version – Provides all available firmware for the current headset version configured on the server side.
10) Get Current Device Last Firmware Version – The client app queries about latest available FW version for a specific headset on the server.
11) Refresh Config Async – The client app updates its internal config from server database.
12) Get SDK Version Async – The client app queries the SDK's Version.

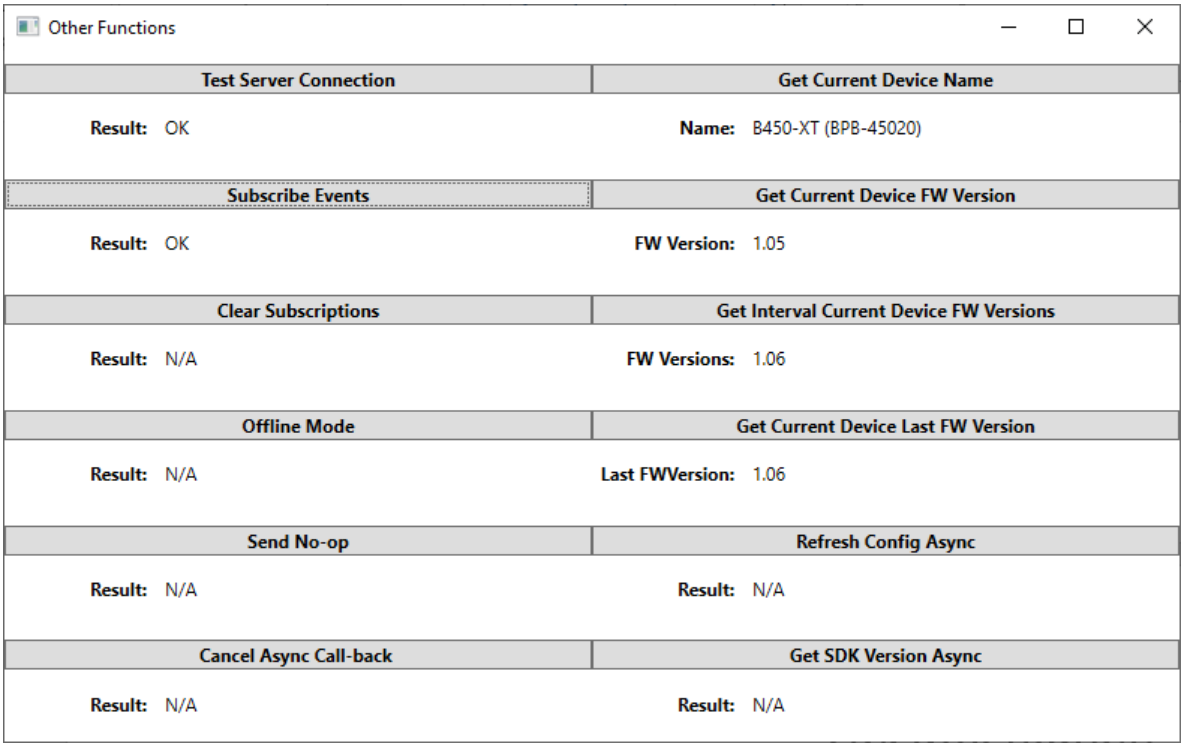

Figure 6. Other functions

Firmware Update Process

The Multi FW Update command is the main feature of the product.

There are some limitations from the QComm libraries and drivers and some limitations in the GNA Headset FW. The limitations are as follows:

1. To update a BP headset, SW has to update all connected BP headsets of the same model in a group update command.
2. SW can only update one group of headsets at a time.
3. All headsets of the same model being updated in a group must all be updated with the same FW image, i.e., to the same FW version.
4. Some headsets have FW version intervals where the headset cannot be updated across intervals boundaries without being updated to a FW version that is an interval relay version.

The client App can navigate limitations 2 and 4 by sequencing several Multi FW Update commands to update all connected headsets to the desired FW version:

1. The client App can update headsets of 2 or more models by issuing one Multi FW Update command for each headset model detected.
2. The client App can update headset FW across a FW intervals by issuing 2 Multi FW Update commands: one to update the headsets to the FW version that is the relay version, then one to update the headsets to the desired FW version.

The Multi FW Update command can command the SDK to update headsets with a FW image from the FW Image Repo Server. In this case the SDK accesses the FW Image Repo Server and downloads the FW image to complete the command. When the SDK downloads a FW image from the FW Image Repo Server it saves that image file locally for further use until the end of that session. During that session if the SDK receives any further Multi FW Update commands to update a headset to that same FW version it uses the saved local file instead of downloading the FW image from the server again.

The update is also possible from a local file (browse button). In this case the end user is responsible for the firmware image.

The update process consists of 3 stages: Reconfiguring, Downloading, Verifying (figures 7,8,9)

There is a slight difference between the 5126 and 8670 chips. 5126 headsets are updated all together. However, 8670-based headsets are updated one by one.

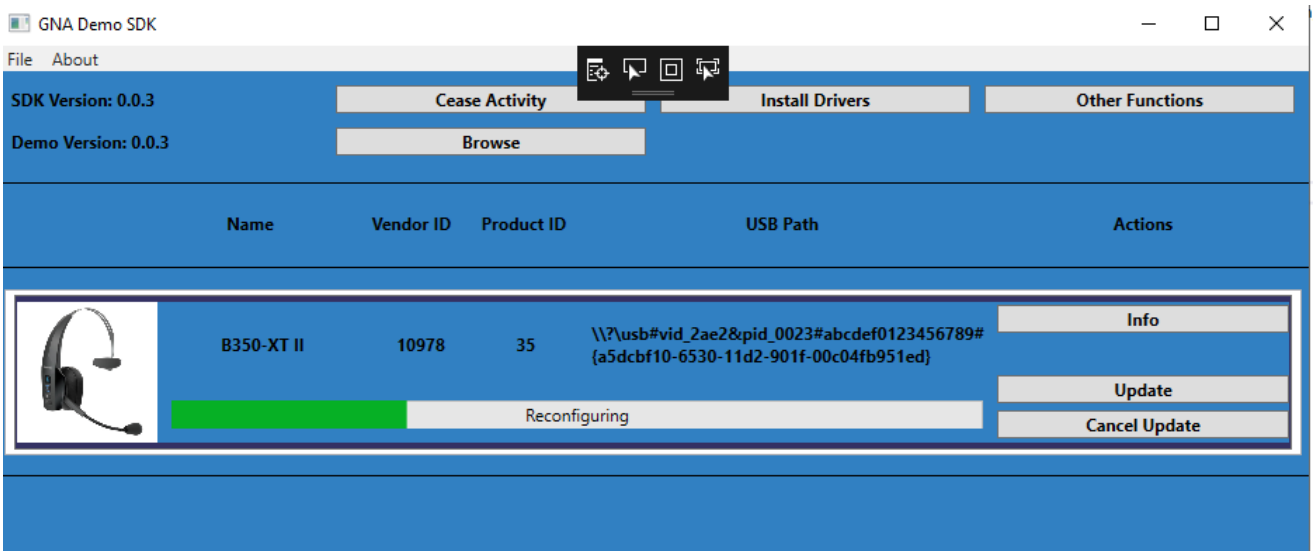

Figure 7. Update Prosses. Reconfiguring.

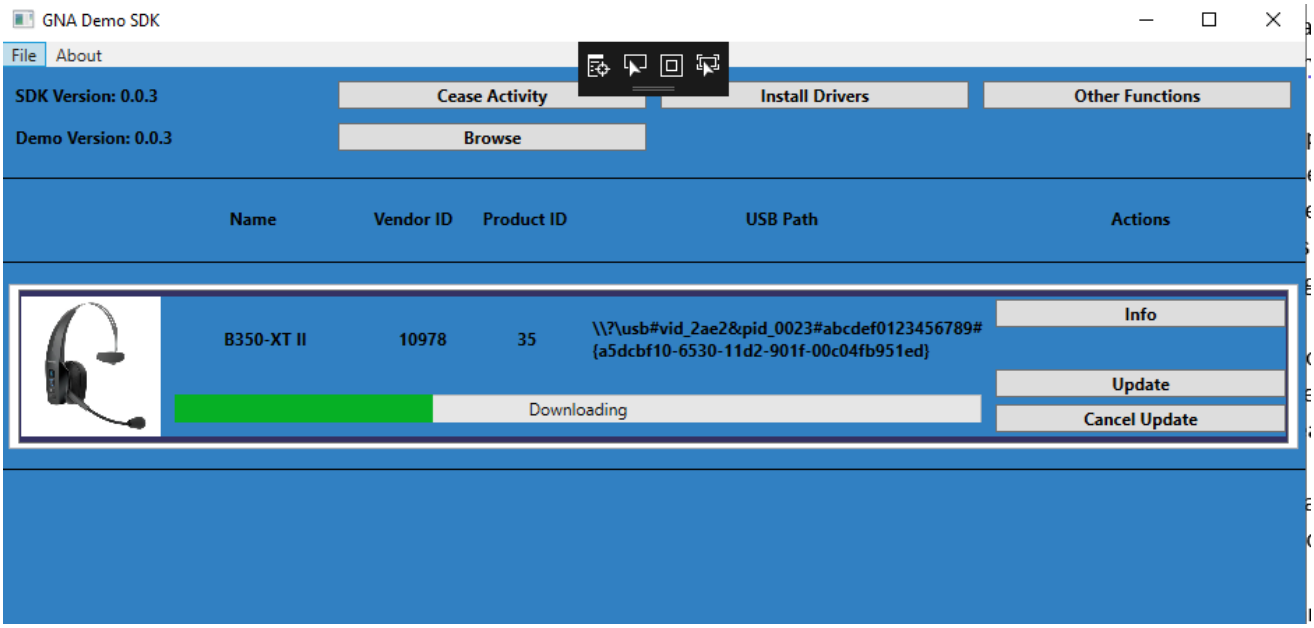

Figure 8. Update Prosses. Downloading.

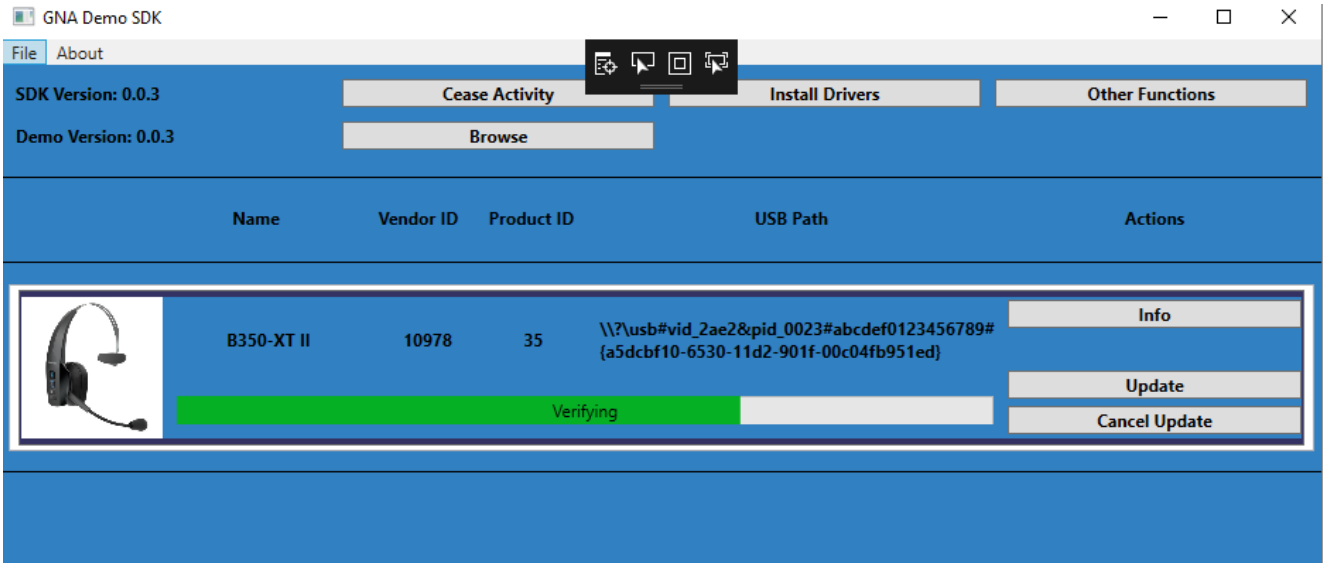

Figure 9. Update Prosses. Verifying.