Documentation

GNA Updater SDK v1.1.0 for Windows

# GNA Updater SDK for Windows

# VERSION: 1.1.0

**Release date: 03-11-2022**

## Scope

This release note applies to GNA Updater SDK software version 1.1.0 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.

Contents

[GNA Updater SDK for Windows 2](#_Toc118473395)

[VERSION: 1.1.0 2](#_Toc118473396)

[Scope 2](#_Toc118473397)

[Software description 2](#_Toc118473398)

[Windows (OS) supported 5](#_Toc118473399)

[System prerequisites 5](#_Toc118473400)

[Device models supported 5](#_Toc118473401)

[Limitations 6](#_Toc118473402)

[GNA Updater SDK 6](#_Toc118473403)

[How the SDK Works 6](#_Toc118473404)

[Including the SDK in the project 6](#_Toc118473405)

[SDK Configurable settings 8](#_Toc118473406)

[Logging path 10](#_Toc118473407)

[SDK API 10](#_Toc118473408)

[Idle State 11](#_Toc118473409)

[Active State 12](#_Toc118473410)

[Background Processing 15](#_Toc118473411)

[Device configuration 16](#_Toc118473412)

[SDK error codes 18](#_Toc118473413)

[Demo Application 25](#_Toc118473414)

[Main Window 26](#_Toc118473415)

[Info Button 27](#_Toc118473416)

[Other Functions 28](#_Toc118473417)

[Device configuration 33](#_Toc118473418)

## 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** |
| Perform 45 | 0x2E45 | 5126 based | USB HID DFU |
| S650-XT / B650-XT | 0x28 | 5126 based | USB HID DFU |
| Talk 65 | 0x2A | 3020 based | USB HID DFU |
| M300-XT | 0x29 | 5126 based | USB HID DFU |
| C300-XT | 0x22 | 8670 | USB QC DFU |
| B550-XT | 0x21 | 8670 | USB QC DFU |
| B250-XT SE | 0x2B | 8670 | USB QC DFU |
| B450-XT II | 0x25 | 8670 | USB QC DFU |
| B350-XT | 0x23 | 8670 | USB QC DFU |
| C400-XT | 0x20 | 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 (see Figure 1)
6. Click OK

Graphical user interface, text, application

Description automatically generated

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 ,2830 | 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 provided within the SDK:

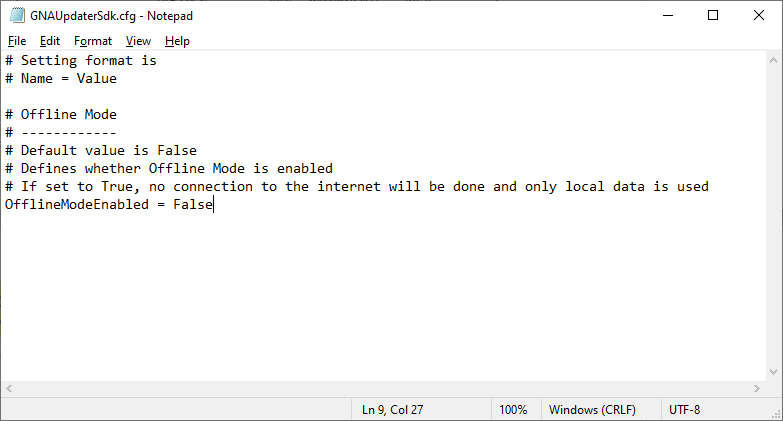


Figure 2. Sample config file

## 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.

|  |
| --- |
| 1. 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   1. public static int **GetSDKVersion**(out string version) – provides the SDK Version string   where:  version – provides the SDK version   1. 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 **GetAvailableFirmwareVersionsForDevice**(int DeviceID, out string[] versions) – provides a list of connected headsets   where:  devices – array of connected devices   1. int **GetSerializedAvailableDevices**(out string serializedDevices) – provides a list of connected headsets   where:  serializedDevices – serialized array of connected devices   1. 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   1. int **GetDeviceName**(int DeviceID, out string deviceName) - provides headset name   where:  DeviceID – device unique identifier  deviceName – headset name   1. int **GetCurrentFirmwareVersion**(int DeviceID, out string deviceFWVersion) – provides current headset firmware version   where:  DeviceID – device unique identifier  deviceFWVersion – current headset firmware version   1. int **GetLatestFirmwareVersion**(int DeviceID, out string deviceFWVersion) – provides latest headset firmware version   where:  DeviceID – device unique identifier  deviceFWVersion – latest headset firmware version   1. int **GetAvailableFirmwareVersionsForDevice**(int DeviceID, out string[] versions) – provides a list of available headset firmware version   where:  DeviceID – device unique identifier  versions – latest headset firmware version   1. int **GetHeadsetImage**(int DeviceID, out byte[] deviceImage) – provides current headset image   where:  DeviceID – device unique identifier  deviceImage – headset image bytes   1. int **InstallDrivers**(InstallDriversCallback callback, ulong context) - installs necessary drivers   where:  callback – invoked when operation is complete  context - 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.   1. int **StartUpgrade**(int DeviceID, UpdateStatusCallback statusCallBack, UpdateDeviceCallback callback, ulong context, string firmwareFile = 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.   1. int **CancelFWUpgrade**() – cancels active update process for the whole headset group 2. 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. 3. int **SubscribeEvents**(AttachedDeviceCallback Attached, DetachedDeviceCallback Detached, DevicePropertiesUpdatedCallback DevicePropertiesUpdated, DBUpdatedCallback DBUpdated = null, DeferredErrorHandler Deferred = null, NoOpCallback NoOp = null) – subscribes callback functions for necessary events   Where:  Attached – invoked when a new headset is connected  Detached – invoked when a headset is disconnected  DevicePropertiesUpdated – invoked when the headset is disconnected  DBUpdated - invoked when the database is updated  Deffered – invoked when the deferred error occurs  NoOp - invoked when SendNoOp API interface is called.   1. int **ClearSubscriptions**() - clearing all subscriptions 2. 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   1. int **CancelAsyncCallback**(ulong context) – cancels asynchronous callback   Where:  context – context value that will be used to cancel Async Call-In Functions. All executing Async Call-In Functions started with given context parameter, will get cancelled.   1. int **TestServerAccess**(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 - 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.   1. 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 - 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.   1. int **QueryConfigurationSettings**(int DeviceID, out string configurationSettings) – provides a data block in JSON format of the config settings for that headset model.   where:  DeviceID - device unique identifier  configurationSettings – serialized list of config settings for given headset model in JSON format   1. int **QueryHeadsetConfiguration**(int DeviceID, QueryHeadsetConfigurationCallback callback, ulong context) – provides a data block in in JSON format of the current config settings for that headset.   Where:  DeviceID - device unique identifier  callback – invoked when operation completes  context - 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.   1. int **SetHeadsetConfiguration**(int DeviceID, string deviceConfiguration, SetHeadsetConfigurationByDeviceCallback deviceCallback, SetHeadsetConfigurationCallback callback, ulong context, bool isReportOnly = false) - applies config settings specified in the data block for every attached headset of that model.   where:  DeviceID - device unique identifier  deviceConfiguration - serialized list of desired config settings for given headset model in JSON format  deviceCallback – invoked when configuration operation completes for individual device  callback – invoked when configuration operation completes for all devices  context - 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.  isReportOnly - decides whether actual configuration happens. If set to true, no configuration takes place, in this mode function can be used to check if device is legible for configuration and if provided configuration string contains valid data. |

### 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.

### Device configuration

1. Retrieving full list of available configuration options with possible values.

In order to retrieve the list of available configuration options, QueryConfigurationSettings API command is used. Returned configurationSettings value will be the JSON string in following format:

|  |
| --- |
| [     {       “name”: “PBConfig”,       “options”: [          {             “name”: “Mute”,             “description”: “Press the Parrott Button™ to mute the microphone. Press again to un-mute.”          },          {             “name”: “Voice Control”,             “description”: “Press the Parrott Button™ to trigger a voice command.”          }       ],       “description”: “Parrott Button Configuration”     },     {       “name”: “Warehouse”,       “options”: [          {             “name”: “On”          },          {             “name”: “Off”          }       ]     }  ] |

Returned JSON value is an array of settings. Each setting has a name and optionally a description and an array of options. Each option has a name and optionally a description.

1. Reading the device configuration:

In order to read current configuration settings for device, QueryHeadsetConfiguration API command is used. Value returned via callback function will be the JSON string in following format:

|  |
| --- |
| [     { "PBConfig" : "Mute" },     { "Warehouse" : "Off" }  ] |

Returned JSON value is an array of key / value pairs. The setting key is the same as the setting name from the QueryConfigurationSettings command. The setting value is the same as the option name from the QueryConfigurationSettings command.

1. Writing the device configuration:

In order to write the configuration to device(s), SetHeadsetConfiguration API command is used. It operates on more than one headset one at a time in a sequence, same as StartUpgrade API command.

For the data in the client specifies a group of headsets of the same model the same way it does so for the StartUpgrade. The client also sends headset a config data block in JSON format similar to the QueryHeadsetConfiguration command.

The limitations are as follows:

1. All devices should have the latest available firmware version
2. Input configuration should have all fields set and all fields should have the valid values (Full list of available settings as well as possible values for them are retrieved via SDK QueryConfigurationSettings API command)
3. SDK should not be in an “Opaque” mode, it should have access either to local or online database.

## 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. |
| -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. |
| -130 | DeviceSettingsReadFailed | Failed to read the device settings for the device. If the problem persists, please report this error to the SDK development team. |
| -131 | DeviceConfigurationReadFailed | Failed to read device configuration option. If the problem persists, please report this error to the SDK development team. |
| -132 | DeviceConfigurationWriteFailed | Failed to write device configuration to device. If the problem persists, please report this error to the SDK development team. |
| -133 | DeviceConfigurationReadInProgress | Another configuration read operation for this device is in progress, please wait for it to complete or cancel the previous operation. |
| -134 | DeviceConfigurationWriteInProgress | Another device configuration write operation is in progress, please wait for it to complete or cancel the previous operation. |
| -135 | DeviceConfigurationInvalidFormat | Failed to validate the configuration string input. Please verify that input is a valid string in a JSON format. |
| -136 | DeviceConfigurationInvalidData | Configuration string input is invalid. Please verify that input complies with the rules described in “Device Configuration” section of this documentation. |
| -137 | DeviceConfigurationInvalidVersion | One or more devices do not have the latest available firmware version installed. Please upgrade your devices before performing the configuration operations. |
| -138 | DeviceConfigurationFailed | Failed to configure all devices of provided model. If the problem persists, please report this error to the SDK development team. |
| -139 | DeviceConfigurationInterrupted | Configuration operation is interrupted by devices of the same model being attached or detached during the operation. |
| -140 | DeviceConfigurationPartialSuccess | Configuration operation has partially succeeded. In case of reading the configuration, one or more settings failed to read, while one or more settings were successfully read. In case of writing the configuration, configuration of one or more devices has failed, while one or more devices has succeeded. |
| -150 | DeviceServiceOperationCancelled | Operation was cancelled by user. |
| -151 | OpenTestEngineFailed | Dependency module error. Unable to get device configuration data. If the problem persists, please report this error to the SDK development team. |
| -152 | FailedToReadDeviceHidData | Failed to read device HID data. If the problem persists, please report this error to the SDK development team. |
| -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 sample 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;
* Read device configuration;
* Write device configuration;

## 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.

1. The device list – provides a device list with available headsets. Each device contains the following info: Name, Vendor Id, Product ID, USB Path
2. Four buttons for each device: Info, Update, Cancel Update

Info – opens up a new window with device information

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)

Configure – opens up a new window for device configuration

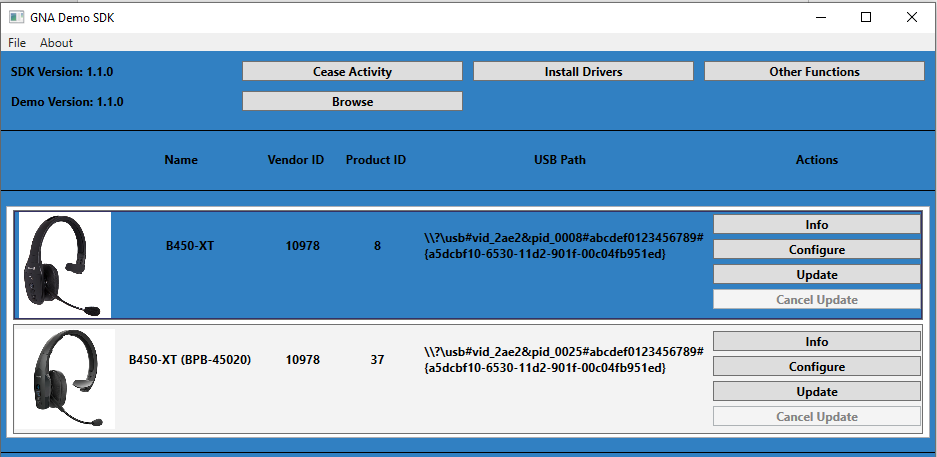


Figure 4. Main Window

## 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

Text

Description automatically generated with low confidence

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.

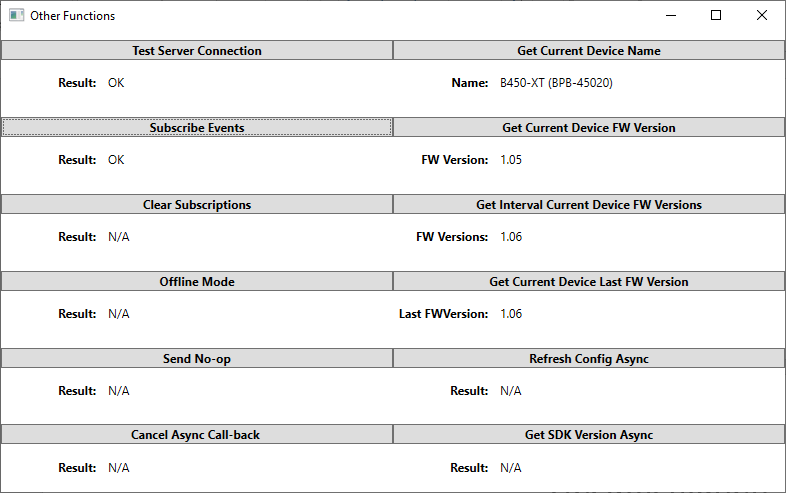


Figure 6. Other functions

**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.

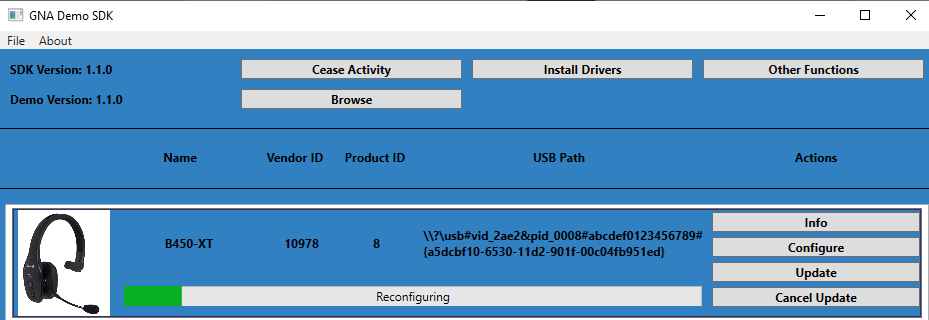


Figure 7. Update Prosses. Reconfiguring.

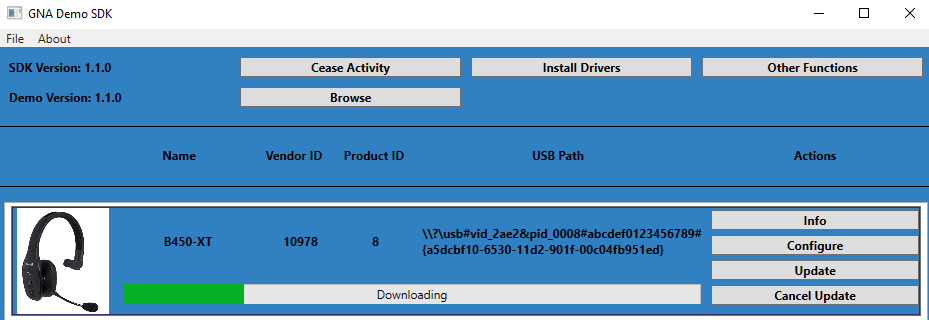


Figure 8. Update Prosses. Downloading.

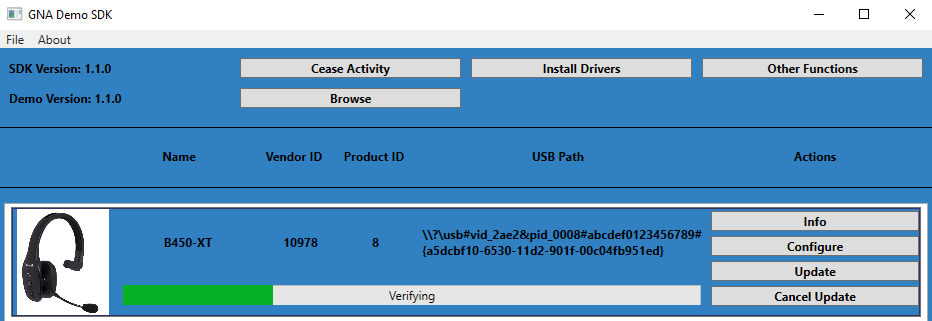


Figure 9. Update Prosses. Verifying.

## Device configuration

Device configuration window features the SDK configuration API that allows to manage device settings. Device settings are defined on the data server and are acquired via web request to that server. Without the internet connection local DB (either provided along with the product or downloaded via last successful internet connection) will be used for configuration data. In an opaque mode however, device configuration is unavailable, and trying to use SDK configuration API will result in an error.

Upon opening the configuration window by clicking the “Configure” button for a specific device from device list, new window will be opened and configuration settings read operation shall start (see. Figure 10).

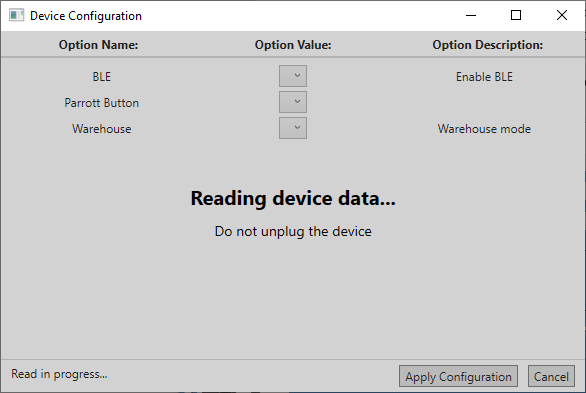


Figure 10. Device configuration options loading

After a successful configuration read operation, configuration window data gets populated with current device data values (see Figure 11).

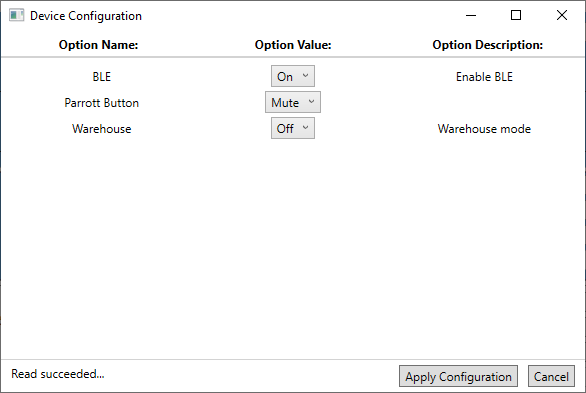


Figure 11. Device configuration options loaded

User is able to change the values of options, by selecting from the dropdown list of available option values.

Upon clicking on “Apply Configuration” button, SDK will try to perform Multi Device Configuration command. Similar to Multi FW Update, it will try to apply the configuration settings for all devices of selected device model.

The limitations are as follows:

1. All devices should have the latest available firmware version
2. Input configuration should have all fields set and all fields should have the valid values (Full list of available settings as well as possible values for them are retrieved via SDK QueryConfigurationSettings API command)
3. SDK should not be in an “Opaque” mode, it should have access either to local or online database.

Clicking “Cancel” button will abandon any unapplied changes and close the configuration window for selected device.