Open Mind	OMNI Summit Microservice V1.X Test Protocols	Doc. Number	Rev.
		OMNI-SM-011	1

Contents

1 Document Purpose

The purpose of this Verification Test Protocol Definition is to describe the testing step-by-step for each requirement specified in the OMNI-SM-002 Software Requirements Specification based on the Software Verification Plan in OMNI-SM-005.

2 Document Scope

This document describes the procedure for performing all Summit Microservice verification testing activities. These software verification test protocols apply to the OMNI Summit Microservice and the software requirements defined in the OMNI-SM-002 Software Requirements Specification. Verification of the OMNI Summit Microservice relies heavily on automated unit and integration testing. The Unit tests are run automatically each time the OMNI Summit Microservice is released, whereas demonstration tests are performed manually or implicitly before release.

Each requirement defined in OMNI-SM-002 Software Requirements Specification is listed in the Software Verification Plan Summary (Appendix 1). Requirements are listed in numerical order by requirement number.

The Software Verification Plan Summary in Appendix 1 includes the following information:

- Software requirement identification number
- Text of the requirement
- Method(s) for verification: Inspection (I), Analysis (A), Demonstration (D), Test (T)

3 Responsibilities

Function	Responsible For
Software Developers	● Develops and executes verification protocols ● Writes protocol-specific reports
Lead Developer	● Writes the verification summary report

4 Automated Unit Testing

4.1 Test Procedure

Automated unit testing is executed on every pull request into the main branch of the Summit Microservice GitHub repository, however it can also be performed manually when cloning the repository to a desktop. Run in either GitHub actions or in visual studio, the passing of a requirement's "Associated Unit Tests" as defined in the below table (4.2) indicates that the requirement is met.

4.2 Capturing of Results

The author of the release's test report will capture the success or failed execution result for each unit test. The unit test result supports each system requirement as indicated below:

Req #	Software Requirement	Associated Unit Tests
4.3	The Microservice shall run in a standalone command console interface, indicating that it is active by writing a start-up message, and allowing for graceful shutdown via keypress or via the exit button on the terminal window.	***SummitServerTest.Startup_CreatesWindow*** confirms that the microservice creates a window allowing for graceful shutdown.

5.1.1	During start-up the microservice shall: ● Initialize a gRPC server with three gRPC-defined services: ○ A device service ○ A bridge service ○ A supporting info service ● Write to the console the current version number and supported devices before entering the ‘Active Mode’.	***SummitServerTest.Startup_PrintsToConsole*** confirms that the console is written with the current version number and supported devices. ***Info_Service_InspectRepository*** confirms that an info service has been created. ***ConnectedBridges_WithBridges_ReturnsBridges*** confirms that a bridge service has been created. ***DeviceStatus_WithSuccessfulRead_ReturnsBatteryStatus*** confirms that a device service has been created.

5.1.2	While in ACTIVE mode, the microservice shall: ● Allow a connected client to request a function to be called on a device or bridge instance. ● If the device or bridge does not exist, the request will be denied. ● If the device or bridge does exist, the function call is forwarded on to the Summit DLL defined instance, which after completion of execution returns the response to the client.	***DeviceStatus_WithSuccessfulRead_ReturnsBatteryStatus*** confirms that a connected client can request a function to be called on a device instance. This also tests that the function call is forwarded on to the Summit DLL, which after completion of execution returns the response to the client. ***DeviceStatus_DeviceDoesNotExist_ReturnsError*** confirms that if the device does not exist, the request is denied. ***DescribeBridge_DoesNotExist_ReturnsError*** confirms that if the bridge does not exist, the request will be denied. ***DescribeBridge_NoErrorFromAPI_ReturnsDetails*** confirms that a connected client can request a function to be called on a bridge instance. This also tests that the function call is forwarded on to the Summit DLL, which after completion of execution returns the response to the client.

5.1.3	When the user enters ‘q’ into the command console or presses the ‘x’ button to close the window, the microservice shall enter shut-down mode. In shut-down mode the microservice shall: ● Safely close all connections to gRPC clients and connections to devices and bridges ● Close the port the microservice was listening on and terminate the process.	***SummitServerTest.Startup_CreatesWindow*** confirms that the microservice creates a window that is shut down by asserting that the process has exited. ***Shutdown_DisposesManager*** confirms that device and bridge connections are disposed when the Server is shut down. ***Startup_CreatesWindow_qToShutdown*** confirms that the server is shut down when ‘q’ is entered into the command console.

5.2.1	The Microservice shall provide an interface for querying the state of the microservice itself through: ● **Version**, which shall return the microservice major, minor, and patch version numbers, and supported devices. ● **SupportedDevices**, which shall return the class of supported implantable neurostimulators (i.e. Medtronic Summit RC+S). ● **InspectRepository**, which shall return all cached device and bridge connections the microservice is currently storing in the in memory repository.	***Info_Service_VersionNumber*** confirms that the microservice shall return the major, minor, and patch version numbers. ***Info_Service_SuppotedDevices*** confirms that the microservice shall return the list of supported implantable neurostimulators. ***Info_Service_InspectRepository*** confirms that the microservice shall return all cached device and bridge connections.

5.2.2	The Microservice shall provide an interface for querying and managing the state of Telemetry Bridges through: ● **ListBridges**, which shall return a list of the known bridges and discover new bridges. ● **ConnectedBridges**, which shall return a list of the bridges currently connected to by the microservice. ● **DescribeBridge**, which shall return the telemetry information of the bridge device	***ListBridges_WithDuplicateBridges_ReturnsUnique, ListBridges_WithQuery_ReturnsFiltered,*** and ***ListBridges_WithNoBridges_ReturnsNoBridges*** confirm that listBridges will return a list of known bridges. Discovery of known bridges is Summit API functionality that is handled through same interface and does not need testing. ***ConnectedBridges_WithBridges_ReturnsBridges, ConnectedBridges_WithQuery_ReturnsFiltered,*** and ***ConnectedBridges_WithNoBridges_ReturnsNoBridges*** confirm that the microservice shall return a list of bridges currently connected to by the microservice. ***DescribeBridge_ErrorFromAPI_ReturnsErrorEmptyDetails*** and ***DescribeBridge_NoErrorFromAPI_ReturnsDetails*** confirm that the microservice shall return telemetry information for a specified bridge

5.2.3	● **ConnectToBridge**, which shall initiate connection to the bridge, and set the number of reconnect attempts before failure. ● **DisconnectFromBridge**, which shall dispose of the device and remove the details of the connection from the repository.	***ConnectToBridge_CachedConnectionIsDisposed _ReturnSuccess*** confirms that when the Summit is in the repo and disposed that the connection call still proceeds. ***ConnectToBridge_CachedConnectionNotDisposed _ReturnsSuccess*** confirms that when the Summit is in the repo but not disposed that the connection call is aborted but success is still returned. ***ConnectToBridge_NoCachedConnectionConnectFails _ReturnsError*** confirms that Summit connection failures are returned to the application. ***ConnectToBridge_NoCachedConnection_ReturnsSuccess _ReturnsSuccess*** confirms that Summit connection success are returned to the application. **ConnectToBridge_NoDiscoveredBridgeRequest _ReturnsError** confirms that an error is returned if an invalid bridge is requested to connect to. ***DisconnectFromBridge_DisposesSummit*** confirms that the DisposeSummit function is called using the mock verify function.

5.2.4	- **ConfigureBeep**, which shall enable telemetry sound state of bridge.	***ConfigureBeep_ValidRequestParameter_ReturnsSuccess*** confirms that the microservice can configure the telemetry beep using API commands. ***ConfigureBeep_InvalidRequest_ReturnsError*** confirms that the microservice returns an error when an invalid parameter is provided to configure the telemetry beep. ***ConfigureBeep_ErrorEncountered_ReturnsFailure*** confirms that the microservice returns an error when the microservice receives an error from the Summit API.

5.2.5	The Microservice shall provide an interface for streaming bridge related status: ● **CreateBridgeConnectionStream,** creates a streaming connection from the bridge connection status updates to be sent to the client application.	***ConnectionStatusStream_NoCachedConnection_ReturnsEmpty*** confirms that an empty (ended) stream is returned if there is no active connection to monitor. ***ConnectionStatusStream_AlreadyConnected_ReturnsEmpty*** confirms that an empty (ended) stream is returned if there is no active connection to monitor. ***ConnectionStatusStream_EnableStreamRequest_ReturnsStream*** confirms that a stream is returned if a request is made while there is a connection and there is no preexisting connection stream. ***ConnectionStatusStream_ConnectedDisableStream_EndsStream*** confirms that a stream can be ended by the client requesting to disable the stream. ***ConnectionMonitoring_DisconnectionEventReceivedSummitNotDisposed_Notify*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a disconnect occurs when the Summit is not disposed. ***ConnectionMonitoring_DisconnectionEventReceivedSummitDisposed_Notify*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a disconnect occurs when the Summit is disposed.

		ConnectionMonitoring_DisconnectionEventReceived_ReconnectionCTMAttemptsExceeded confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a CTM reconnection attempt fails. ***ConnectionMonitoring_DisconnectionEventReceived_ReconnectionINSAttemptsExceeded*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when an INS reconnection attempt fails. ***ConnectionMonitoring_DisconnectionEventReceived_ReconnectionSucceeded*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a reconnection suceeds.

5.2.6	The Microservice shall provide an interface for querying the state of INS devices through: ● **ListDevices**, which shall return a list of devices connectable by a given bridge. ● **DeviceStatus**, which shall return the battery status of the device connected ● **LeadIntegrityTest**, which shall return the impedance values between given pairs of electrodes	***ListDevices_WithNoDevices_ReturnsNoDevices*** confirms that if no devices are discovered by CTM that no devices are returned to the client. ***ListDevices_WithSameDeviceConnectedAlready _ReturnsMedtronicError*** confirms that if a device is already connected to, that the expected Medtronic error code is returned to the client. ***ListDevices_WithDeviceSuccess_ReturnsDeviceList*** confirms that if a device is discovered that it is returned to the client. ***DeviceStatus_WithSuccessfulRead_ReturnsBatteryStatus*** confirms that on a successful device status call, that battery status information is returned to the client. ***DeviceStatus_DeviceDoesNotExist_ReturnsError*** confirms that if the device does not exist that an error is returned to the client. ***DeviceStatus_WithUnsuccessfulRead _ReturnsNullBatteryLevels*** confirms that if an error is encountered during the device status request that an error is returned to the client. ***LeadIntegrityTest_NoError_ReturnsLeadList*** confirms that the lead integrity test returns impedances if successful. ***LeadIntegrityTest_SameLeadValues_ReturnsZero*** confirms that the lead integrity test returns 0 if a selected lead pair contains two of the same electrode.

		LeadIntegrityTest_OutofRangeLeadValues_ReturnsError confirms that the lead integrity test returns out of range error if invalid lead indexes are provided.
5.2.7	The Microservice shall provide an interface for querying and managing the state of INS devices through: ● **ConnectToDevice**, which shall establish a connection between the device, bridge, and the server. ● **DisconnectFromDevice**, which shall disband a connection between a given device, bridge, and the server.	***ConnectToDevice_RequestedDeviceNotCached_ReturnsError*** confirms that when the requested device is not in the device repo, that the Summit connection is aborted are returned to the application. ***ConnectToDevice_NoDeviceSpecified_NullConnectSuccess*** confirms that if no device is specified in the request, that a null device parameter is provided to the Summit StartInsSession command to initiate a null-connect. ***ConnectToDevice_CachedDeviceConnect_ReturnsConnectStatus*** confirms that the Summit connection StartInsSession command status is returned to the application **ConnectToDevice_NoConnectedBridge_ReturnsError** confirms that an error is returned if GetConnectionByName returns null. ***DisconnectFromDevice_DisposesSummit*** confirms that the DisposeSummit function is called using the mock verify function.

5.2.8	● **SenseConfiguration**, which shall configure all sense settings. The configurable settings shall include: ○ Time Domain Channel Configuration ○ Fourier Transform Channel Configuration ○ Power Channel Configuration ○ Linear-Discriminant Classifier Configuration ○ Accelerometer Configuration ○ Miscellaneous Summit Configuration ○ Sense State Enabling	***ConfigureSense_NullRequestParameter_ReturnsFailure*** confirms that the microservice returns an error if there is a null parameter provided for part of the sensing configuration. ***ConfigureSense_PowerBandEnablesCountIncorrect_ReturnsFailure*** confirms that an error is returned if the power channel list is too long. ***ConfigureSense_NullRequestParametersObject_ReturnsFailure*** confirms that if an entire sensing configuration object (FFT) is null that an error is returned. ***ConfigureSense_WriteSensingFftSettingsReject_ReturnsFailure*** that if a command is rejected as part of the sense configuration transaction that the expected summit API reject is returned to the client. **ConfigureSense_DisabledTimeDomainStream_SetsSampleRateToDisabled** confirms that if a time domain stream is indicated to be disabled that the correct enumeration for disabled channels is transmitted to the Summit API. ***ConfigureSense_ValidConfiguration_Success*** confirms that the microservice returns success when no error is encountered in the sense configuration transaction.

5.2.9	The Microservice shall provide an gRPC interface to provide data streaming through the following commands: ● **StreamEnable**, which shall enable streaming capabilities from the device, through the bridge, to the client application. ● **StreamDisable**, which shall disable streaming capabilities from the device, through the bridge, to the client application. ● **TimeDomainStream**, which shall create a stream for Time Domain data updates to be sent from the device, through the bridge, to the client application. ● **FourierTransformStream**, which shall create a stream for Spectral data updates to be sent from the device, through the bridge, to the client application.	***StreamEnable_ValidRequestParameter_ReturnsSuccess*** confirms that the microservice can enable streaming using API commands. ***StreamEnable_ErrorEncountered_ReturnsFailure*** confirms that the microservice returns an error when parameters are valid but the microservice receives an error from the Summit API ***StreamEnable_InvalidRequestParameter_ReturnsUnknown*** confirms that the microservice returns Unknown when an invalid parameter is provided to enable sensing. ***StreamDisable_ValidRequestParameter_ReturnsSuccess*** confirms that the microservice can disable streaming using API commands. ***StreamDisable_ErrorEncountered_ReturnsFailure*** confirms that the microservice returns an error when parameters are valid but the microservice receives an error from the Summit API ***StreamDisable_InvalidRequestParameter_ReturnsUnknown*** confirms that the microservice returns Unknown when an invalid parameter is provided to enable sensing. For each streaming data type (Time Domain, Fourier, BandPower, Inertial, Adaptive, Loop Record, and Echo) the following four tests are defined which evaluates specific streaming functionality:

	- **BandPowerStream**, which shall create a stream for Band Power data updates to be sent from the device, through the bridge, to the client application. - **InertialStream**, which shall create a stream for Inertial data updates to be sent from the device, through the bridge, to the client application. - **AdaptiveStream**, which shall create a stream for Adaptive Status updates to be sent from the device, through the bridge, to the client application. - **LoopRecordUpdateStream**, which shall create a stream for Loop Recorder Status updates to be sent from the device, through the bridge, to the client application.	***Create<DataType>StreamStream_NoCachedConnection_ReturnsEmpty*** confirms that an error is returned to the client if there is no connected device to stream from. ***Create<DataType>StreamStream_AlreadyConnected_ReturnsEmpty*** confirms that an error is returned to the client if the specific data type stream is already active. ***Create<DataType>StreamStream_EnableStreamRequestData_ReturnsStream*** confirms that a stream is returned to the client if there is a connected device that is not actively streaming the requested specific data type. ***Create<DataType>StreamStream_ConnectedDisableStream_EndsStream*** confirms that an active stream can be stopped at the client's request.

	- **EchoStream**, which shall create a stream for Echo data updates to be sent from the device, through the bridge, to the client application.	

5.3.1	When a connected Summit System throws a disconnected event, the Microservice shall begin to send a battery status request every 10 seconds to monitor the connectivity status of the bridge and device. If the Summit System indicates that the connection is not automatically recoverable, the microservice shall then: ● Send a message to connected clients via the bridge connection status streaming endpoint. ● Make an attempt to reconnect to the device automatically up to a specified number of retries. ● Abort the connection attempts after the specified number of retries are exceeded.	***ConnectionMonitoring_DisconnectionEventReceivedSummitNotDisposed_Notify*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a disconnect occurs when the Summit is not disposed. ***ConnectionMonitoring_DisconnectionEventReceivedSummitDisposed_Notify*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a disconnect occurs when the Summit is disposed. ***ConnectionMonitoring_DisconnectionEventReceived_ReconnectionCTMAttemptsExceeded*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a CTM reconnection attempt fails. ***ConnectionMonitoring_DisconnectionEventReceived_ReconnectionINSAttemptsExceeded*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when an INS reconnection attempt fails. ***ConnectionMonitoring_DisconnectionEventReceived_ReconnectionSucceeded*** confirms that the microservice provides an interface for streaming bridge related status that notifies an application when a reconnection suceeds.

6.1.1	The microservice shall allow multiple connected clients to initiate gRPC-enabled functionality utilizing bridge/device name pairs.	***MultipleConnections_TwoClientOneDevice_ConnectionRequestsSuccess*** and ***MultipleConnections_TwoClientTwoDevices_ConnectionRequestsSuccess*** confirm that multiple clients can successfully communicate with microservice-connected devices.
6.1.2	The microservice shall allow each streamed data type to be sent to a single client.	***Create<DataType>StreamStream_AlreadyConnected_ReturnsEmpty*** confirms that an error is returned to the client if the specific data type stream is already active.
6.2.1	The microservice shall support at least two Summit Systems to be simultaneously communicated with by client applications.	***MultipleConnections_TwoClientTwoDevices_ConnectionRequestsSuccess*** and **MultipleConnections_OneClientTwoDevices_ConnectionRequestsSuccess** confirm that the microservice allows for two simultaneous connections to Summit devices.
8.4.1	The microservice shall default all network listening interfaces to be localhost only unless overwritten at the user's discretion. It shall be the user's responsibility to ensure that HIPAA related information is secure in the case where this is overwritten with a non-localhost only address.	***ListBridges_WithDuplicateBridges_ReturnsUnique()*** confirms that without any configuration, localhost is used to host the microservice.

10.4	The messages, endpoints, enums, etc. in the protobuf interface files shall be documented in a format that allows for auto generation of documentation.	Generation of signed interface control document (ICD) demonstrates completion of this requirement.

5 Demonstration

All demonstrations require the following procedure to set up the demonstration environment:

- Download prerequisite software
  - Visual Studio https://visualstudio.microsoft.com/downloads/
  - Git https://git-scm.com/download/win
- Download and set up OmniSummitMicroservice
  - On a computer running windows 10, open a powershell window and navigate to the location you would like to install to. Tutorial
  - Clone from the OmniSummitDeviceService git repository using the following command:

```
git clone --recurse-submodules https://github.com/openmind-consortium/OmniSummitDeviceService
```

  - Document the git short hash by navigating into the cloned folder and entering the following command:
    - git rev-parse --short HEAD
  - Open Visual Studio and select open existing project. Navigate to where you installed OmniSummitDeviceService and select the OmniSummitDeviceService.sln solution file to open the project in Visual Studio.
  - Make sure you have access to the Medtronic Summit RDK
  - Copy the DLL files from the Summit RDK/DLLs/AnyCPU/ to the Libraries folder inside the OmniSummitDeviceService solution, in OmniSummitDeviceService/Libraries/
- Download and set up a Client Application

Python Client Example:

Installation of Client

- Run the following command to download the GitHub repository
  - git clone git@github.com:openmind-consortium/Omni-Python-Examples.git
- Notice the submodule OmniProtos. This contains the gRPC protobuf files that will help this Python client communicate with the OpenMind Server (which is written in C#).
- There are two main files in the repository: client.py and build_protos.sh. client.py is the Python script containing the code for this example. build_proto.sh is a Bash script to build the protobuf files found in the OmniProtos submodule.

Building the Protos

- This example depends on gRPC protobuf files, which need to be built before the code is run. build_protos.sh is a Bash script that can be used to build the protobuf files on a Linux machine (or on Windows Subsystem for Linux [WSL]). To run it on those systems use the command:

```
./build.protos.sh
```

- For other operating systems, make a directory at path/to/github/repo/protos and run the following Python command as described in the gRPC Python documentation:

```
python -m grpc.tools.protoc -IOmniProtos --python_out=./protos --grpc_python_out=./protos
OmniProtos/*.proto
```

- This protos folder is imported as a Python module into client.py. To make the protos directory a proper Python module, create an empty file inside called __init__.py.

Starting the OpenMind Server

- Open the OpenMind Server project in VisualStudio. Build and run it. A terminal window will appear showing that the server is running on localhost:50051.

Running the Python Client

- Now it's time to run client.py. There is one optional flag for an ip address of the OpenMind Server. If not provided, the default location is localhost. However, if the server is running on a different IP address, it can be specified using the flag as follows:

```
python client.py --ip <ip_address>
```

    - Now the application is running. Terminal output will print as the bridge and device are connected, and as sensing is configured, finally resulting in the time domain packet data being printed in a continuous stream. The printing will stop when the program is stopped.
- Connect to a Medtronic Summit System
    - Document the serial numbers of the CTM and Summit RC+S

JavaScript Client Example:

Overview

The OMNI Reference Client is a desktop application built using JavaScript Electron.js Library to interface with OMNI compatible device services. The OMNI client provides a way, through use of the OMNI Summit Microservice, to find and connect to up to two Medtronic Summit RC+S systems, configure data streams, record data, and troubleshoot connections.
The order of operations is as follows:

1. Connect to the OpenMind Server
2. Find and connect to one or two bridges
3. Find and connect to one or two devices
4. Configure sensing on the Summit RC+S
5. Stream time domain data from the Summit RC+S

Pre-requisites

Node.js

First, install node js using the instructions on https://nodejs.org/en/

Obtaining protos files

Github

If the repository was cloned from github, git already linked the OmniProtos files as a submodule. The original repository for OmniProtos can be found here (https://github.com/openmind-consortium/OmniProtos.git). To obtain those files, first clone the OmniReferenceClient repository:

```
git clone https://github.com/openmind-consortium/OmniReferenceClient.git
```

Then, move to the cloned directory and initialize the submodule:

```
git submodule init
```

and update the submodule:

```
git submodule update
```

Zip file

If the repository was obtained using a zip file. create a new folder in the home directory of the repository called protos and copy the .protos files from the protobuff deliverable into that folder.

Sense Configuration Files

In the packaged version of the app, the sensing config files will not be bundled with the app and needs to be copied into this directory on windows: /AppData/Roaming/omniconfig. Examples of these files can be found in the config folder of this repository.

config.json - main config file, the name field of the left and right objects need to be updated with the serial number of both the CTM and the INS: "//summit/bridge//device/".
senseLeft_config.json - sensing config for the left INS, make sure that this file is in the same directory as config.json.
senseRight_config.json - sensing config for the right INS, make sure that this file is in the same directory as config.json.

Architecture

OMNI client is built using electron. The application is broken into three separate processes: renderer, main and preload. The renderer process handles the user interface. The main process interfaces with the backend via gRPC. To do this, a library called grpc-js is installed with the dependencies. In the main process, bridgeClient and deviceClient are called when needed to interface with the backend. For example, deviceClient.ListDevices function is called to retrieve device information from the backend.

The preload acts as a security layer between the main and renderer process. The config.forge section of package.json contains the actual configuration for all of the processes.

The user interface is built using React. Communication between the renderer process and the main process uses the Electron Context Bridge to mitigate security risks exposed by using Electron's ipcRenderer directly in the renderer process.

The OMNI device service is a stateless backend (with the exception of connection state management). Similarly, the main process has no state. The main process acts as a passthrough from the renderer to the OMNI device service. All states are managed by the React application.

The entrypoint for the renderer process is /src/renderer/index.tsx, the entrypoint for preload is /src/preload/index.ts and the entrypoint for the main process is /src/main/index.ts.

Getting Started

Starting the OpenMind Server

Open the OpenMind Server project in VisualStudio. Build and run it. A terminal window will appear showing that the server is running on localhost:50051.

Running the application locally

First, clone repository if not already done so:

```
git  clone https://github.com/openmind-consortium/OmniReferenceClient.git
```

Make sure that the OmniProtos and sense configuration files are set up as explained above. Next, navigate in the terminal to the source folder and install the node dependencies:

```
npm install
```

Next, start the development server:

```
npm start
```

Electron forge does not support hot-reloading. To reload the OMNI client after you've made changes to the source type rs and hit enter from the terminal where you started the development server.

To package the application run:

```
npm run make
```

To lint the code run:

```
npm run lint
```

To have the linter fix errors, run:

```
npm run lint -- --fix
```

To edit the application, use either github to clone the repository (https://github.com/openmind-consortium/OmniReferenceClient) or open the zip file. In the protos folder, copy the google protobuff files included in the package. Make sure dependencies needed to run javascript and Electron are installed in the computer, including but not limited to node.js, and open a command prompt at the root directory.

Req #	Software Requirement	Associated Demonstration Protocol
4.2	The Microservice shall interface with Summit System via the Summit DLLs: Medtronic.SummitAPI.dll, Medtronic.TelemetryM.dll, Medtronic.NeuroStim.Olympus.dll	1) A tester will run the microservice connected to a Medtronic Summit System and verify that they are connected.
7.1	The microservice shall run on Windows 10 with 64 bit depth	1) A tester will download the microservice executable and run it on a Windows 10 64 bit PC. 2) The tester will take a screenshot of the console window running on the PC and upload to the test report document.
7.2	The microservice shall support gRPC clients using version gRPC 1.0.0 or greater.	1) A tester will run the microservice using gRPC version 1.0.0 or greater and verify that after following all of the setup steps above that the application runs successfully and connects to the CTM and Summit. 2) The tester will document in the test report what version of gRPC the client application utilized when the demonstration was executed. This version has to be greater than 1.0 for the test to be considered a success.

7.3	The microservice shall serialize messages using protobuf version 3.	1) A tester will verify that messages are serialized using protobuf version 3. The tester can verify this by checking the version of protobuf 2) The tester will document in the test report what version of protobuf the client application utilized when the demonstration was executed. This version has to be greater than 3.0 for the test to be considered a success.
9.1	Installer	1) A tester will verify that the provided installer creates the OmniSummitMicroservice binary and verifies that the Medtronic Summit DLLs are in the correct location.
10.1	Version Numbering	1) A tester will look over the version history and verify that it follows proper semantic versioning
10.2	Packaging Labeling	1) A tester will verify that the installer's version reflects the version of the microservice
10.3	Documentation Generation	1) A tester will verify that there is a github action in place to update and automatically deploy documentation for each new release

6 Inspection

6.1 Test procedure

There are no requirements that are tested via inspection.

7 References

Document Identifier	Title
OMNI-SM-002	Software Requirements Specification, OMNI Summit Microservice

8 Approvals

Approver Role	Signature and Date
Project Lead	DocuSigned by: David Borton 8A588B8FA1F246E... 11/11/2022 | 1:20 PM EST
Quality	DocuSigned by: D41D8EC9725B4EA... 11/14/2022 | 2:39 AM EST
Lead Developer, Author	DocuSigned by: Jeffrey Herron 64503D12F815426... 11/13/2022 | 11:26 PM PST

9 Appendix 1: Software Verification Summary

Requirement text is for reference only. The SRS is the definitive source for requirement content. Specific protocols referenced are in Appendix 2.

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
4.1	Application Interface	The Microservice shall include an application programming interface (API) based on gRPC. This is defined in the OMNI Summit Microservice Interface Control Document (ICD) v1.x.	T	8 Appendix 2.1 Automated Test
4.2	Device Interface	The Microservice shall interface with a Medtronic Summit System via the Summit DLLs: Medtronic.SummitAPI.dll, Medtronic.TelemetryM.dll, Medtronic.NeuroStim.Olympus.dll	D	8 Appendix 2.3 Inspection

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
4.3	Console Interface	The Microservice shall run in a standalone command console interface, indicating that it is active by writing a start-up message, and allowing for graceful shutdown via keypress or via the exit button on the terminal window.	T	8 Appendix 2.1 Automated Test
5.1.1	Start-up Mode	During start-up the microservice shall: ● Initialize a gRPC server with three gRPC-defined services: ○ A device service ○ A bridge service ○ A supporting info service ● Write to the console the current version number and supported devices before entering the 'Active Mode'.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.1.2	Active Mode	While in ACTIVE mode, the microservice shall: ● Allow a connected client to request a function to be called on a device or bridge instance. ● If the device or bridge does not exist, the request will be denied. ● If the device or bridge does exist, the function call is forwarded on to the Summit DLL defined instance, which after completion of execution returns the response to the client.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.1.3	Shut-Down Mode	When the user enters 'q' into the command console or presses the 'x' button to close the window, the microservice shall enter shut-down mode. In shut-down mode the microservice shall: ● Safely close all connections to gRPC clients and connections to devices and bridges ● Close the port the microservice was listening on and terminate the process.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.2.1	Support Commands	The Microservice shall provide an interface for querying the state of the microservice itself through: ● **Version**, which shall return the microservice major, minor, and patch version numbers, and supported devices. ● **SupportedDevices**, which shall return the class of supported implantable neurostimulators (i.e. Medtronic Summit RC+S). ● **InspectRepository**, which shall return all cached device and bridge connections the microservice is currently storing in the in memory repository.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.2.2	Bridge Query Commands	The Microservice shall provide an interface for querying and managing the state of Telemetry Bridges through: ● **ListBridges**, which shall return a list of the known bridges and discover new bridges. ● **ConnectedBridges**, which shall return a list of the bridges currently connected to by the microservice. ● **DescribeBridge**, which shall return the telemetry information of the bridge device	T	8 Appendix 2.1 Automated Test
5.2.3	Bridge Management Commands	● **ConnectToBridge**, which shall initiate connection to the bridge, and set the number of reconnect attempts before failure. ● **DisconnectFromBridge**, which shall dispose of the device and remove the details of the connection from the repository.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.2.4	Bridge Configuration Commands	● **ConfigureBeep**, which shall enable telemetry sound state of bridge.	T	8 Appendix 2.1 Automated Test
5.2.5	Bridge Connection Status Streaming	The Microservice shall provide an interface for streaming bridge related status: ● **CreateBridgeConnectionStream,** creates a streaming connection from the bridge connection status updates to be sent to the client application.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.2.6	Device Query Commands	The Microservice shall provide an interface for querying the state of INS devices through: ● **ListDevices**, which shall return a list of devices connectable by a given bridge. ● **DeviceStatus**, which shall return the battery status of the device connected ● **LeadIntegrityTest**, which shall return the impedance values between given pairs of electrodes	T	8 Appendix 2.1 Automated Test
5.2.7	Device Management Commands	The Microservice shall provide an interface for querying and managing the state of INS devices through: ● **ConnectToDevice**, which shall establish a connection between the device, bridge, and the server. ● **DisconnectFromDevice**, which shall disband a connection between a given device, bridge, and the server.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.2.8	Device Configuration Commands	● **SenseConfiguration**, which shall configure all sense settings. The configurable settings shall include: ○ Time Domain Channel Configuration ○ Fourier Transform Channel Configuration ○ Power Channel Configuration ○ Linear-Discriminant Classifier Configuration ○ Accelerometer Configuration ○ Miscellaneous Summit Configuration ○ Sense State Enabling	T	8 Appendix 2.1 Automated Test

5.2.9	Device Data Streaming Commands	The Microservice shall provide an gRPC interface to provide data streaming through the following commands: ● **StreamEnable**, which shall enable streaming capabilities from the device, through the bridge, to the client application. ● **StreamDisable**, which shall disable streaming capabilities from the device, through the bridge, to the client application. ● **TimeDomainStream**, which shall create a stream for Time Domain data updates to be sent from the device, through the bridge, to the client application. ● **FourierTransformStream**, which shall create a stream for Spectral data updates to be sent from the device, through the bridge, to the client application. ● **BandPowerStream**, which shall create a stream for Band Power data updates to be sent from the device, through the bridge, to the client application. ● **InertialStream**, which shall create a stream for Inertial data updates to be sent from the device, through the bridge, to the client application.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
		● **AdaptiveStream**, which shall create a stream for Adaptive Status updates to be sent from the device, through the bridge, to the client application. ● **LoopRecordUpdateStream**, which shall create a stream for Loop Recorder Status updates to be sent from the device, through the bridge, to the client application. ● **EchoStream**, which shall create a stream for Echo data updates to be sent from the device, through the bridge, to the client application.		

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
5.3.1	Summit System Connection Monitoring	When a connected Summit System throws a disconnected event, the Microservice shall begin to send a battery status request every 10 seconds to monitor the connectivity status of the bridge and device. If the Summit System indicates that the connection is not automatically recoverable, the microservice shall then: ● Send a message to connected clients via the bridge connection status streaming endpoint. ● Make an attempt to reconnect to the device automatically up to a specified number of retries. ● Abort the connection attempts after the specified number of retries are exceeded.	T	8 Appendix 2.1 Automated Test
6.1.1	Multiple clients to communicate with Summit Systems	The microservice shall allow multiple connected clients to initiate gRPC-enabled functionality utilizing bridge/device name pairs.	T	8 Appendix 2.3 Inspection

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
6.1.2	Each stream can communicate to single endpoint	The microservice shall allow each streamed data type to be sent to a single client.	T	8 Appendix 2.1 Automated Test
6.2.1	Microservice will support two Summit Systems Simultaneously	The microservice shall support at least two Summit Systems to be simultaneously communicated with by client applications.	T	8 Appendix 2.3 Inspection
7.1	The Microservice shall run on Windows	The microservice shall run on Windows 10 with 64 bit depth.	D	8 Appendix 2.2 Demonstration

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
7.2	The Microservice shall support gRPC	The microservice shall support gRPC clients using version gRPC 1.0.0 or greater.	D	8 Appendix 2.2 Demonstration
7.3	The Microservice shall serialize messages using protobuf	The microservice shall serialize/deserialize all outgoing/incoming messages using protobuf version 3.	D	8 Appendix 2.2 Demonstration
8.4.1	The Microservice shall use a localhost-only default address	The microservice shall default all network listening interfaces to be localhost only unless overwritten at the user’s discretion. It shall be the user’s responsibility to ensure that HIPAA related information is secure in the case where this is overwritten with a non-localhost only address.	T	8 Appendix 2.1 Automated Test

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
9.1	Installer	The microservice shall be distributed via an installer program. The installer program shall install the microservice binary, as well as ask for location of Summit RC+S DLLs for the microservice to use.	D	8 Appendix 2.2 Demonstration
10.1	Version Numbering	The microservice shall follow semantic versioning (i.e. 1.2.3 where 1 is major version, 2 is minor version, 3 is patch version, or 1.0.0-beta for a beta release of version one. More information can be found: https://semver.org/spec/v2.0.0.html). The version shall be displayed during microservice start-up, and shall be available via the version endpoint.	D	8 Appendix 2.2 Demonstration
10.2	Packaging Labeling	The installer's version shall reflect the version of the microservice.	D	8 Appendix 2.2 Demonstration

Req #	Software Requirement Heading	Software Requirement	Method (IADT)	Protocol Number
10.3	Documentation Generation	The documentation shall be updated and deployed automatically with each new release.	D	8 Appendix 2.2 Demonstration
10.4	Protobuf Documentation	The messages, endpoints, enums, etc. in the protobuf interface files shall be documented in a format that allows for auto generation of documentation.	T	8 Appendix 2.1 Automated Test

10 Appendix 2: Test protocol definitions

Automated Test

Automated tests are run as part of every merge into the main branch. If any of these tests fails, the branch cannot be merged back into the main branch. This allows us a high confidence that the OMNI Summit Microservice is running correctly. SOUP behavior is mocked so automated tests can more easily test different code paths.

Demonstration

Certain requirements are demonstrated by the ability to run automated tests. If the automated tests failed, these demonstration tests would also fail. Demonstration test examples include “The microservice supports Windows 10.”

Inspection

Some requirements require inspection of build artifacts.