웹소켓 통신 기반

W3C 자율주행차 차량정보 접근 인터페이스 표준해설서

머리말

2020년 신종 코로나바이러스감염증-19(COVID-19) 사태로 언택트(비대면) 기술 확산과 함께 산업의 중심이 오프라인에서 온라인으로 옮겨지면서 ICT의 중요성이 더욱 커지고 있습니다. 정부는 포스트 코로나 시대에 대응하여 '한국판 뉴딜' 정책을 발표로 '디지털 뉴딜', '그린 뉴딜', '안정망 강화'의 대형 프로젝트를 진행하여, D.N.A 생태계 강화, 비대면 사업육성, SOC 디지털화 등 코로나 이후의 위기를 극복하기 위해 디지털 국가로의 발전 정책을 추진 중에 있습니다.

D.N.A(Data-Network-AI)를 기반으로 디지털 국가와 스마트한 정부, 국토, 산업을 목표로 더 안전하고 편리해진 국민들의 삶 조성을 위하여 데이터댐, AI, 스마트의료, 디지털트윈(Digital Twin) 등의 ICT 기반 기술 발전과제를 선정하였습니다. ICT 표준은 R&D와 시장을 연결하는 핵심수단으로, ICT 분야의 기술적 상용화와 사용자 수요에 맞는 기술 개발을 위한 ICT 표준화를 추진하고 있습니다.

한국정보통신기술협회는 ICT 국제표준화전문가 활동지원을 통해 국내 산업체의 ICT 시장 경쟁력을 제고하고 국제표준화기구에서의 한국 영향력 확대를 위해 노력하고 있습니다. ICT 표준화포럼에서는 국내 산업체 등 중소기업의 수요를 반영하여 IETF의 I2NSF(Interface to Network Security Functions), W3C의 RTC(Real-time Communication Between Browsers), DID(Decentralized ID), Vehicle cloud(Vehicle Information Service Specification)의 최신 표준 4개를 선정하고, 관련 표준의 해설을 담은 ICT국제표준화전문가 표준해설서를 마련하였습니다.

본 표준해설서는 해당 기술의 구현 및 지식습득이 필요한 국내 산업체와 ICT국제표준전문가 표준화 활동에 교과서적인 전문적 지식전달 목적으로 활용될 수 있을 뿐만 아니라, ICT 사실표준화의 국내 산업 활성화에 기여할 수 있을 것으로 기대합니다. 마지막으로 표준해설서 발간에 참여해 주신 사실표준화기구 미러포럼 표준전문가 여러분과 중소·중견기업 관련자 분께 깊이 감사드립니다.

목차

1. 표준 개요 ····· 1

2. W3C 및 Automotive WG 소개 ····· 2

1) W3C(World Wide Web Consortium) 소개 ····· 2

2) Automotive WG 소개 ····· 3

3. Vehicle Information Service Specification 유즈케이스 ····· 5

1) 차량 모니터링 및 관리 서비스 ····· 5

2) 자율주행차-스마트홈 연동 서비스 ····· 6

4. VISS(Vehicle Information Service Specification) 표준 해설서 ····· 7

1) W3C Vehicle Information Service Specification 표준 ····· 8

(1) 소개(Introduction) ····· 8

(2) 적합성(Conformance) ····· 13

(3) 용어(Terminology) ····· 13

(4) 그림표(Table of Figures) ····· 13

(5) 아키텍처(Architecture) ····· 13

(6) 보안 및 개인 정보 고려 사항(Security and Privacy Considerations) · 19

(7) Websocket 초기화(Initialisation of the WebSocket) ····· 22

(8) 메시지 구조(Message Structure) ····· 24

(9) Server Side Filtering ····· 72

(10) WebSocket 종료(WebSocket Closure) ····· 75

(11) 오류(Errors) ····· 75

(12) 참고문헌(References) ····· 78

2) Vehicle Information Service Specification 해설 ····· 79
(1) 표준 1장 소개 부분에 대한 해설 ····· 79
(2) 표준 2장 적합성(Conformance)에 대한 해설 ····· 80
(3) 표준 3장 용어(Terminology)에 대한 해설 ····· 80
(4) 표준 4장 그림표(Table of Figures)에 대한 해설 ····· 80
(5) 표준 아키텍처(Architecture)에 대한 해설 ····· 80
(6) 표준 보안 및 개인 정보 고려 사항(Security and Privacy Considerations)에 대한 해설 ····· 81
(7) 표준 Websocket 초기화(Initialisation of the WebSocket)에 대한 해설 ····· 82
(8) 표준 메시지 구조(Message Structure)에 대한 해설 ····· 82
(9) 표준 9장 서버측 필터링(Server Side Filtering)에 대한 해설 ····· 84
(10) 표준 10장 WebSocket 종료(WebSocket Closure)에 대한 해설 ····· 85
(11) 표준 11장 오류(Errors)에 대한 해설 ····· 85
(12) 참고문헌(References)에 대한 해설 ····· 85

1. 표준 개요

차량 정보 서비스 표준(Vehicle Information Service Specification)은 자율주행차의 내부나 외부의 응용이나 서비스가 차량 정보를 접근해 사용할 수 있는 표준 인터페이스에 대한 사양이다. 본 표준은 기본적으로 HTML5의 양방향 통신 방법인 웹소켓 통신을 기반으로 하고 있으며, 실제적인 Request/Response는 JSON 기반의 메시지를 사용한다. 에러 코드의 경우는 HTTP의 에러코드를 확장 정의하여 기존의 웹 개발자들이 쉽게 이해하고 활용할 수 있도록 하였다. 또한 인증 메커니즘은 OAuth를 사용하여 Access Token을 기반으로 한다.

2. W3C 및 Automotive WG 소개

1) W3C(World Wide Web Consortium) 소개

1989년 최초 개발 이후, 웹의 급속한 발전으로 인하여, CERN이 전체 망과 관련 표준기술을 독자적으로 개발하는 것은 불가능하게 됨에 따라, 웹의 발명자인 Tim Berners-Lee는 DARPA (U.S. Defense Advanced Research Project Agency)와 European Commission으로부터 지원을 받아 1994년 10월 미국 MIT(Massachusetts Institute of Technology)에서 "웹의 모든 잠재력을 이끌어 내기 위하여(Leading the Web to its Full Potential)"라는 모토 하에 모든 웹의 가능성 개발 및 향후 진화를 위한 기술적인 가이드 제시를 목적으로 W3C(World Wide Web Consortium)를 설립하였으며, 창립된 이래 민간 비영리단체로써 웹과 관련 기술들에 대한 표준화를 주도하고 있다.

현재 W3C는 구글, 애플, 삼성전자, LG전자, 페이스북, MS, Oracle, AirBnB 등을 포함하여 전세계적으로 429개의 회원사가 참여하고 있으며, 국내에서는 ETRI, 삼성전자, LG전자, DRM 인사이드, 구루미, 잉카인터넷, 인스웨이브시스템, KETI, KIPFA, 온페이스SDC, 리모트모스터, SCE korea와 같이 12개사가 참여하고 있다.

W3C 표준화 과정은 그림 1-1과 같이 Working Draft, Candidate Recommendation, Proposed Recommendation의 단계를 거쳐 최종 표준 단계인 Recommendation으로 구성된다.

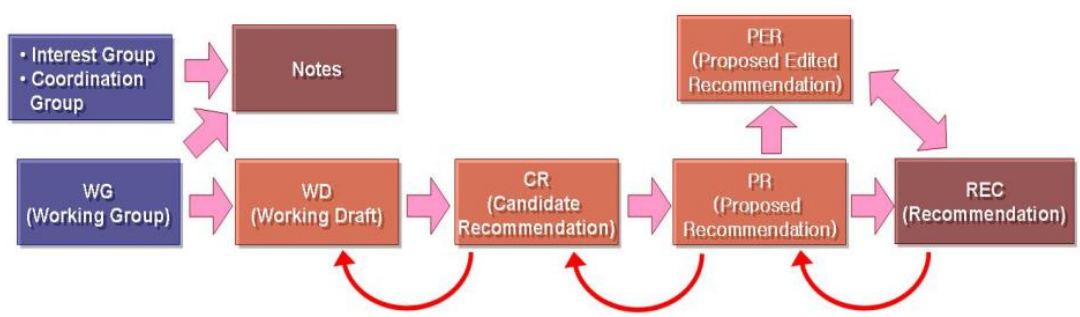


그림 1-1. W3C 표준 권고안 제정 절차

(1) WD(Working Draft) 단계

○ W3C의 모든 그룹과 외부에 최초 Working Draft 발간을 공지하고, 최초 Working Draft 발간은 문서 검토가 시작됨을 알리는 것으로서, 특히 특허 정책과 관련하여 기준이 되는 시점이 됨. 이후 WG은 해당 WD 문서를 지속적으로 보완하여, 제기된 문제들을 해결하고 수정된 부분은 반드시 표시해야 하며, 워킹그룹은 WD 단계 동안 팀, 회원, 다른 그룹, 외부의 모든 검토를 받아야 함

(2) CR(Candidate Recommendation) 단계

○ WD 단계에서 CR단계로 전환을 요청하는 경우, W3C 디렉터는 자문위원회(AC)에 구현 요청(Call for Implementations)을 공지하며, 이를 통해 안정적이고 구현이 가능한 상태임을 검증함. 구현 경험을 바탕으로 해당 문서는 변경될 수 있음

(3) PR(Proposed Recommendation) 단계

○ CR 단계에서 모든 구현 검증이 끝난 경우, WG에서는 PR단계로의 전환을 요청하게 되며, Director가 자문위원회에 Proposed Recommendation에 대한 검토를 요청함. PR 단계의 검토 결과를 통해 표준 초안 문서에 대한 안정성과 W3C 회원사들의 지지를 확인하게 됨

(4) REC(Recommendation) 단계

○ PR단계가 마무리되면 Director는 자문위원회에 W3C 권고안을 공지함으로써 표준 제정이 선포됨을 알림

2) Automotive WG 소개

HTML5 표준 개발을 주도하는 W3C(World Wide Web Consortium)는 2012년부터 자동차 관련 기업들이 참여하고 있는 GENIVI 얼라이언스와 협력하여 오토모티브 웹(Automotive Web) 표준을 개발을 시작하였다. 2012년 2월에 W3C는 GENIVI 얼라이언스와 적극적인 협력을 위해 오토모티브 및 웹 플랫폼 BG(Business Group)를 설립하였고, 2015년 2월에 오토모티브 워킹그룹(Automotive WG)을 설립하고 본격적인 자율주행차를 위한 표준 개발을 시작하였다.

Automotive WG에는 19개의 회사에서 64명이 전문가가 참여하고 있다. 국내에서는 ETRI, LG전자, OnfaceSDC가 참여 중이며 해외에서는 폭스바겐 자동차그룹, 볼보, 재규어 랜드로버, GeoTab 등이 참여하고 있다.

현재 Automotive WG에서는 Vehicle Information Service Specification 표준안 개발에 대해 마무리 단계에 있어 2020년 말 또는 2021년 초에 W3C 최종 표준으로 제정할 예정이다. 또한 차기 표준으로 Vehicle Information Service Specification – Version2를 개발 중에 있으며, 본 표준에서는 RESTFul 인터페이스로의 확장과 차량 데이터 필터링 기능의 확장 등 추가적인 기능에 대한 표준 개발을 진행하고 있다.

3. Vehicle Information Service Specification 유즈케이스

자율주행차 시대가 되면 기본적으로 차량이 인터넷에 연결되어 있어야 하며, 이때 차량과 차량 내외부의 응용, 서비스 등과 연동을 통한 다양한 유즈케이스가 존재한다. 이러한 유즈케이스의 실현을 위해 차량 데이터를 접근하기 위한 인터페이스 표준인 Vehicle Information Service Specification이 필수적으로 요구된다.

1) 차량 모니터링 및 관리 서비스

그림 3-1. 차량 인터페이스를 활용한 다양한 서비스 시나리오

차량 데이터를 접근할 수 있는 표준 인터페이스를 활용한 가장 기본적인 유즈케이스는 차량에 대한 모니터링 및 관리 서비스이다. 기본적으로 B2B 관점에서는 쏘카, 카카오 택시 등 기업들은 현재 운영하는 차량의 위치 및 차량의 상태에 대한 모니터링이 필요하다. 즉, 차량이 방치되어 있지는 않은지 또는 차량의 엔진오일이나 타이어 공기압 등 차량의 상태에 문제가 없는지 확인이 필요하다. 이러한 정보를 기반으로 효율적으로 차량을 관리할 수 있을 뿐 아니라 데이터 분석을 통해 기업의 수익율을 극대화할 수 있다. 개인 운전자 측면에서는 차량의 상태정보를 자신이 자주 이용하는 카센터와 연계하여 안전하게 차량을 관리할 수 있다.

2) 자율주행차-스마트홈 연동 서비스

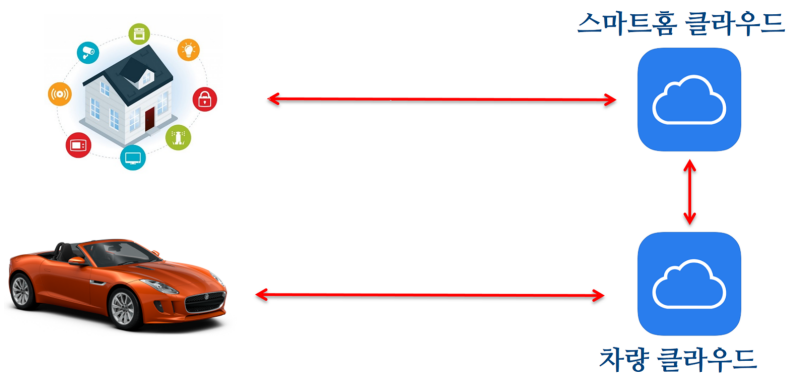


그림 3-2. 자율주행차와 스마트홈 연동

그림 3-2와 같이 Vehicle Information Service Specification 표준을 기반으로 자율주행차와 스마트홈 연동에 활용이 가능하다. 예를 들면 겨울이라고 가정하고 차량이 집을 목적지로 이동하고 있다고 생각해 보자. 차량이 집에 도착하기 30분전에 차량은 스마트홈에 히팅(난방)을 요청할 수 있을 것이다. 또한 아침에 회사로 출근할 때는 스마트홈에 저전력 모드로의 설정을 요청할 수도 있고, 때로는 청소와 환기 등 정리를 요청할 수도 있을 것이다. 다양한 사용자의 생활 패턴을 분석하여 사용자를 돕기 위한 자동화를 지속적으로 최적화하는 자율주행차와 스마트홈 연동 시나리오로 생각해 볼 수 있다.

4. VISS(Vehicle Information Service Specification) 표준 해설서

- 이 문서의 버전:
  https://www.w3.org/TR/2018/CR-vehicle-information-service-20180213/
- 발행된 최신 버전:
  https://www.w3.org/TR/vehicle-information-service/
- 편집자 최신 원고(draft):
  https://w3c.github.io/automotive/vehicle_data/vehicle_information_service.html
- 브라우저 구현 현황(Implementation report):
  https://www.w3.org/auto/wg/wiki/VISS_implementations
- 이전 버전:
  https://www.w3.org/TR/2016/WD-vehicle-information-service-20161020/
- 편집자:
  Kevin Gavigan
  Jaguar Land RoverAdam Crofts
  Jaguar Land Rover
  이원석(Wonsuk Lee) 한국전자통신연구원(ETRI)
  Powell Kinney
  Vinli
- 표준 참여하기:
  We are on github.
  File a bug/issue.
  Commit history.
  Mailing list archive.

1) W3C Vehicle Information Service Specification 표준

본 표준은 2018년 2월 13일에 공개된 Vehicle Information Service Specification에 대한 내용이며, 지금까지 논의되었던 이슈들이 거의 정리되어 2020년 말이나 2021년 초에 W3C 최종 권고안으로 공개될 것으로 예상된다.

초록(Abstract)

본 사양은 차량 정보 서비스를 위한 웹소켓(WebSocket) 기반 API를 정의하여 클라이언트 애플리케이션이 차량 신호(vehicle signal) 및 데이터 속성을 접근할 할 수 있도록 한다.

사양의 목적은 참여하는 자동차 제조업체 간에 일관된 방식으로 애플리케이션 개발을 가능하도록 지원하기 위한 서버 API를 확산하기 것이다.

(1) 소개(Introduction)

이 섹션은 비표준(non-normative)이다.

본 사양은 차량이 웹소켓을 통해 차량 신호 및 정적 데이터를 제공하는 방법을 설명한다. 이를 통해 클라이언트는 차량 신호 및 정적 데이터를 GET 또는 SET 할 수 있다. 차량정보에 대한 알림을 수신하려면 SUBSCRIBE을 사용하고, 알림 수신을 해제하려면 UNSUBSCRIBE를 사용한다.

차량 신호 및 정적 데이터에 대한 액세스를 관리하는 데 사용되는 API는 본 차량 정보 서비스 사양(VISS)에 정의한다. VISS는 또한 클라이언트가 특정 시점에 접근할 수 있는 차량 신호 및 데이터의 세트를 확인하는 검색 메커니즘을 설명한다.

여기서 '신호'라는 용어는 차량 속도와 같이 시간에 따라 변할 수 있는 데이터 항목을 나타내기 위해 사용되며, 반면 '정적 데이터' 또는 단순히 '데이터'라는 표현은 차량 길이와 같은 변하지 않는 속성을 갖은 프로퍼티(property)를 표현하는 데 사용된다.

차량 기반 소프트웨어 모듈은 본 사양에 정의된 인터페이스 및 동작을 구현할 수 있다. 이에 대한 구현은 WebSocket 서버 인스턴스를 생성하는 것으로 이는 클라이언트의 인바운드(inbound) 연결 요청을 수신하고, 차량 신호에 대한 안전한 접근을 가능하게 한다. 본 사양에서는 이 모듈을 VIS 서버(VIS Server) 또는 '서버(the server)'라고 한다.

본 사양은 데이터 모델에 확실히 독립적으로 차량 데이터에 접근하기 위한 여러 방법을 정의한다. 문자열을 사용하여 데이터와 신호를 지정할 수 있는 모든 데이터 모델은 잠재적으로 지원될

수 있다. 그러나 이 버전의 사양은 데이터 모델이 GENIVI Vehicle Signal Specification [VSS]임을 지정한다. VSS는 확장성과 개인 브랜치(private branch)를 정의하는 기능을 모두 지원하며, 본 사양의 각 예제에 사용된다.

또한 언제든 접근할 수 있는 신호의 '트리(tree)'는 표준 접근 제어 원칙에 따라 달라질 수 있다. 즉, 데이터를 요청하는 사용자(개인 또는 조직)의 신원 및/또는 요청이 발생한 응용 또는 장치(예: 차량)에 따라 달라질 수 있다.

이를 지원하기 위해 VIS 사양은 선택적으로 토큰을 VIS 서버에 전달하는 데 사용할 수 있는 확장 가능한 토큰 기반 보안 메커니즘을 설명한다. 예를 들어 응용의 사용자 및/또는 클라이언트 응용이 실행 중인 장치를 나타낸다.

본 사양의 향후 개정에서는 RESTful 웹 서비스를 통해 추가적인 차량 신호를 지원할 것을 고려할 것이며 이는 본 사양의 범위를 벗어난다.

다음 예제는 설명 목적으로만 사용되며 오류 처리를 보여주지 않으며 상업용 코드가 아니다.

EXAMPLE 1

```
// Open the WebSocket
var vehicle  = new WebSocket("wss://wwwivi", "wvss1.0");

// Establish authorization
vehicle.onopen = function () {
  vehicle.send('{"action": "authorize", "tokens": {"authorization": "user_token_value"},
"requestId": "103"}');
};

// Message response handler for all possible actions
vehicle.onmessage = function(event){
  var msg = JSON.parse(event.data);
  switch(msg.action){
    case "authorize":
      authHandler(msg);
      break;
    case "getMetadata":
      getMetadataHandler(msg);
      break;
    case "get":
      getReqHandler(msg);
      break;
    case "set":
      setReqHandler(msg);
      break;
    case "subscribe":
      subscriptionSetupHandler(msg);
      break;
    case "subscription":
      subscriptionHandler(msg);
      break;
    case "unsubscribe":
      unsubscribeHandler(msg, false);
      break;
    case "unsubscribeAll":
      unsubscribeHandler(msg, true);
      break;
  }
};

// Auth request handler
function authHandler(msg) {
  if(msg.hasOwnProperty("TTL")){
    console.log("authorization successful");
    requestMetadata();
  } else {
    console.log("authorization unsuccessful");
  }
}

// Close the WebSocket to end the WebSocket session
function closeSocket(){
  vehicle.close();
}
```

WebSocket 연결이 설정되면 클라이언트는 서버에서 메타데이터(metadata)를 검색하고 클라이언트의 현재 인증 레벨에 대한 신호를 요청할 수 있다.

EXAMPLE 2

```
// Request the entire data model, in this example the data model is defined using GENIVI's VSS
function requestMetadata(){
  vehicle.send('{"action": "getMetadata", "requestId": "104"}');
}

// Request the entire data model and if successful, request a signal
function getMetadataHandler(msg){
  if(msg.hasOwnProperty('metadata')){
    console.log("Metadata Received");
    console.log("Metadata: " + JSON.stringify(msg.metadata));
    getSignal(msg.metadata, "Signal.Drivetrain.Transmission.Speed");
  } else {
    console.log("getMetadata Error");
  }
}

// Request a signal
function getSignal(metadata, path){
  // A check could be made here to ensure the signal is available within the metadata
  vehicle.send('{"action": "get", "path": ' + path + ', "requestId": "105"}');
}
```

클라이언트는 또한 클라이언트의 인증이 접근을 허용하는 경우 subscribe action을 이용하여 신호를 주기적으로 받을 수 있다.

EXAMPLE 3

```
// Set a subscription, assuming the same authorization and set up from Example 1
var rpmSubscriptionId, rpmRequestId = "106";

// Set up the subscription
function subscribeToRPMNotifications(){
  vehicle.send('{"action": "subscribe", "path":
"Signal.Drivetrain.InternalCombustionEngine.RPM",  "requestId": ' + rpmRequestId + '}');
}

// Handle the subscription response
function subscriptionSetupHandler(msg){
    if(msg.hasOwnProperty("requestId") && msg.requestId == rpmRequestId){
      console.log("Subscription set with a subscription ID of " + msg.subscriptionId);
      rpmSubscriptionId = msg.subscriptionId;
    }
}

// Handle the subscription notification
function subscriptionHandler(msg){
    if(msg.hasOwnProperty("subscriptionId") && msg.subscriptionId == rpmSubscriptionId){
      console.log("The current engine rpm is " + msg.value);
    }
}
```

본 사양에서 지원하는 대상 플랫폼은 승용차로 국한한다. 비승객(non-passenger) 응용(예: 중장비, 해양 및 항공 인포테인먼트)에서 본 사양을 사용하는 것은 금지되지 않지만, 본 사양의 API의 설계에서 이들에 대한 고려를 하지 않았다.

유즈케이스의 예로 타이어 압력, 엔진 오일 수준, 와셔액 수준 및 배터리 충전 수준과 같은 주의가 필요한 경우 경고를 제공하는 'Home Mechanic' 응용을 구현할 수 있다.

차량 신호를 표준화된 방식으로 제공하면 사물웹(Web of Things) 내에서 차량 통합이 용이해지고 교통, 안전, 내비게이션, 에너지 관리, 스마트 운송 및 소비자 인포테인먼트를 비롯한 다양한 분야에서 미래의 혁신적인 유즈케이스를 지원할 수 있다.

본 사양에서 클라이언트/서버 통신에 사용되는 데이터 형식은 JSON 인코딩 문자열이지만 향후 사양 버전에서 다른 데이터 형식이 지원될 수 있다.

(2) 적합성(Conformance)

비표준(non-normative)으로 표시된 섹션 뿐만 아니라 본 사양의 모든 작성지침, 다이어그램, 예제 및 노트는 비표준(non-normative)이다. 본 사용에서 이들을 제외한 다른 모든 내용은 표준이다.

키워드 MAY, MUST, SHALL은 [RFC2119]에 설명된 것과 같이 해석되어야 한다.

본 사양은 단일 제품에 적용되는 적합성 기준을 정의한다: 특히 이 문서에 정의된 인터페이스, 의미 및 동작을 구현한 '차량 내' WebSocket Vehicle Information Service.

(3) 용어(Terminology)

약어 'VISS'는 본 문서인 'Vehicle Information Service Specification'을 지칭하는 데 사용된다.

약어 'VSS'는 GENIVI Alliance에서 정의한 'Vehicle Signal Specification'을 지칭하는 데 사용된다.

본 문서에서 사용되는 'WebSocket'이라는 용어는 W3C WebSocket API 및 WebSocket 프로토콜에 정의된 것과 같다.

(4) 그림표(Table of Figures)

Figure 1 Diagram showing an example Vehicle Signal Tree

Figure 2 Diagram showing relationships between the vehicle system, VIS Server and its clients.

Figure 3 State diagram for the VIS Server

Figure 4 Diagram showing conceptual WebSocket Security Token flow

(5) 아키텍처(Architecture)

이 섹션은 비표준(non-normative)이다.

① 개요

일반적인 차량 설계에서 신호와 데이터는 내부 차량 네트워크를 통해 연결된 ECU(Electronic Control Unit)간에 전송된다. 여기에는 CAN(Controller Area Network), MOST(Media Oriented Systems Transport) 및 LIN(Local Interconnect Network)이 포함된다. 이러한 네트워크의 ECU는 네트워크 버스에서 메시지를 브로드캐스트하고 버스의 다른 ECU는 메시지에 응답한다.

Figure 1에 포함된 구성요소 다이어그램에서 내부 차량 CAN, MOST 및 LIN 네트워크와 이러한 네트워크를 통해 통신하는 ECU는 추상화되고 단순성을 위해 System 컴포넌트(System component)로 표시된다.

안전, 보안 및 상업적 이유로 모든 클라이언트가 특정 차량 신호 및 데이터 속성을 GET, SET 또는 SUBSCRIBE 할 수 있는 것은 아니다. 결과적으로 접근 제어를 관리하여 클라이언트가 단순히 ECU 또는 CAN, MOST 또는 LIN 네트워크 버스에 직접 연결할 수 없도록 해야 한다.

차량 시스템 내에서 사용 가능한 차량 신호 및 정적 데이터는 제어된 방식으로 VIS 서버에 노출된다. 차량 시스템과 VIS 서버 간에 사용되는 인터페이스 및 통신 메커니즘은 본 사양의 범위에 포함되지 않는다.

② VIS 서버(Vehicle Information Service Server)

위에서 언급했듯이 VIS 서버는 Vehicle Information Service Specification(VISS)에 정의된 인터페이스 및 동작을 구현하여 차량 신호 및 데이터를 온보드(on-board) 클라이언트에 노출하는 '차량 내' 시스템이다.

VIS 서버는 함께 제공되는 데이터 모델과 일치하는 방식으로 신호 및 데이터를 노출한다. 데이터 모델은 본 사양에서 제한하지 않지만, 본 사양에서 사용하고 권장하는 데이터 모델은 Vehicle Signal Specification(VSS)이다. VSS는 차량의 '트리와 같은(tree-like)' 논리적 텍사노미(공식적으로 Directed Acyclic Graph)를 정의한다. 이는 주요 차량 구조(예: body, engine)가 트리의 상단 근처에 있고 이를 구성하는 논리적 어셈블리 및 구성 요소가 자식 노드로 정의된다. 트리의 각 자식 노드는 논리적 구성 요소로 더 분해되고 리프 노드(leaf nodes)에 도달할 때까지 프로세스가 반복된다. 리프 노드는 단일 신호 또는 데이터 속성 값을 나타내기 때문에 분해할 수 없는 분기의 끝에 있는 노드이다. VSS 트리의 예는 Figure 2와 같으며 본 내용은 예시이다.

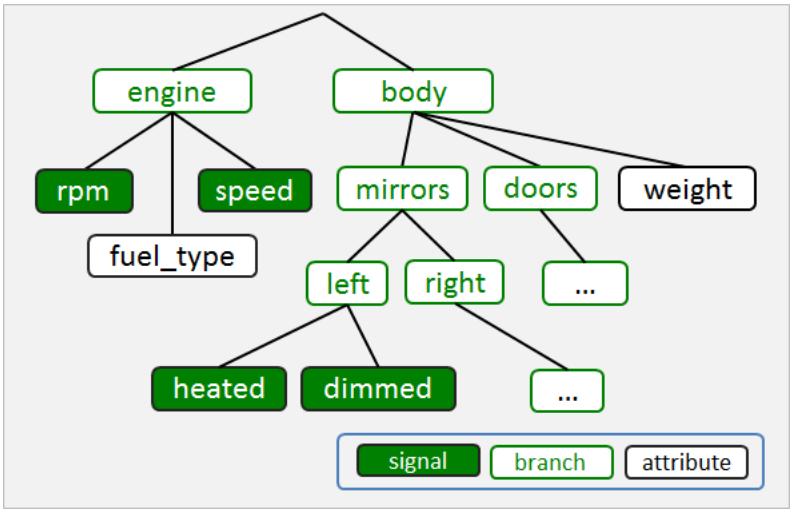


Figure 1. 차량 신호 트리의 예를 보여주는 다이어그램

GENIVI의 VSS 데이터 모델 내에서 신호는 점 표기법(dot notation)(예: engine.rpm)을 사용하여 경로에 따라 이름이 지정된다. 본 문서에 정의된 메서드는 문자열로 표현할 수 있는 모든 경로 표기법(path notation)을 지원한다.

클라이언트는 VIS 서버의 getMetadata 메서드를 호출하여 잠재적으로 접근 가능한 신호 및 데이터의 속성 정보를 기술한 메타데이터를 서버가 반환하도록 요청할 수 있습니다. 물론 이것은 사용자 및/또는 장치가 적절하게 권한을 부여받은 경우에 가능하다. 이 액션(action)과 기타 유효한 VIS 서버 액션(actions) 은 여기에 좀 더 자세히 정의되어 있다.

③ 차량 내 클라이언트(In-Vehicle Clients)

차량 내 클라이언트에는 두 경우의 클라이언트가 모두 포함됩니다. 하나는 차량 자체의 ECU에서 실행되는 클라이언트로 차량내 인포테인먼트(IVI) 시스템에서 구현되며, 다른 하나는 사용자의 디바이스(예: 노트북, 휴대폰 또는 태블릿)에서 실행되는 클라이언트로 이는 차량의 WiFi가 있는 경우 이를 통해 연결된다.

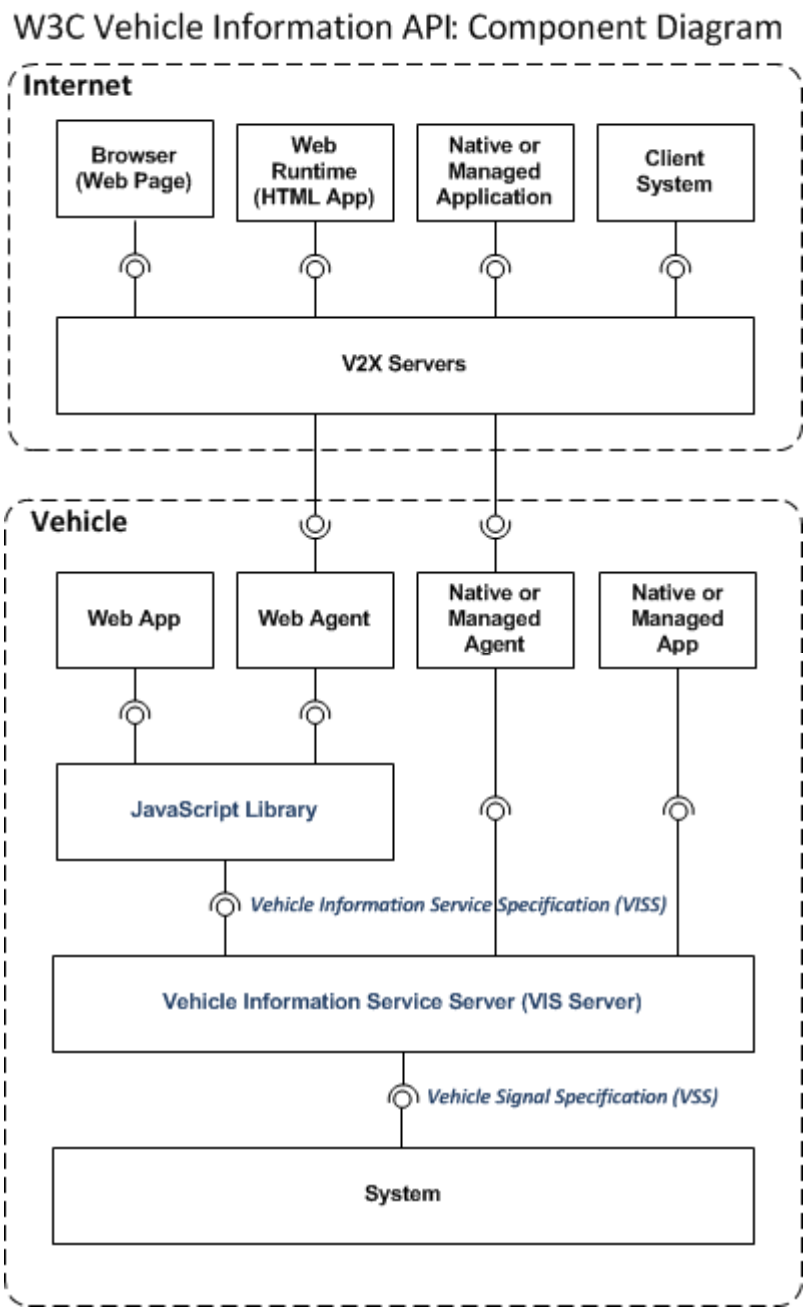


Figure 2. 차량 시스템, VIS 사버 및 해당 클라이언트 간의 관계를 보여주는 다이어그램

VIS 서버 인터페이스에 정의된 메서드는 다양한 유형의 온보드(on-board) 및 오프 보드(off-board) 클라이언트에서 호출할 수 있다. 차량에서 실행되는 온보드 클라이언트는 다음 두 가지 주요 범주 중 하나에 속한다:

○ 응용 프로그램(Applications) - 차량 신호 및 데이터에 접근할 수 있고 차량의 운전자 또는 승객과 상호 작용을 위한 사용자 인터페이스가 있는 차량에서 실행되는 응용

○ 에이전트(Agents) - 일반적으로 에이전트에는 사용자 인터페이스가 없다. 또한 VIS 서버에서 제공하는 메서드를 호출할 수 있지만 그 목적은 하나 이상의 오프 보드(off-board)(예: 데이터를 외부로 전송하기 위한 V2X(Vehicle to Everything) 인터넷 서버)에 연결하는 것이다.

응용 프로그램과 에이전트는 모두 다음과 같이 세분화될 수 있다. (ⅰ) 웹기반은 '웹 런타임'에서 실행되며 HTML, CSS 및 JavaScript와 같은 웹 표준을 사용하여 구현될 수 있다. (ⅱ) 그들 자신의 네이티브 프로세스에서 실행되도록 구현될 수 있다. 이 경우는 예를 들어 C, C++ 또는 Objective-C로 구현된다. (ⅲ) 또한 C# 또는 Java와 같은 언어 구현된 관리 런타임 프로세스 실행되도록 구현될 수 있다.

웹 애플리케이션 및 웹 에이전트는 VIS 서버에 의해 제공되는 차량 정보 서비스 인터페이스를 직접 호출하거나 JavaScript 'wrapper' 라이브러리를 구현하여 제어된 방식으로 차량 신호 및 데이터 액세스를 단순화할 수 있다.

④ 신호 및 데이터의 오프보드 전송(Sending Signals and Data off-board)

로컬, 차량 내 애플리케이션 및 에이전트 외에도 다양한 인터넷 기반 클라이언트 및 서버가 차량 신호 및 데이터에 액세스하는 데 관심이 있을 수 있다. 그러나 차량을 사용하지 않을 때는 배터리 수명을 최대화하기 위해 대부분의 전기 시스템이 차단된다. 여기에는 무선 또는 모바일 연결을 활성화하는 시스템이 포함될 수 있다. 차량의 전원이 켜지면 오프 보드 연결을 제공하는 시스템을 포함하여 다양한 시스템이 시작되고 차량은 일반적으로 로컬 WiFi 네트워크 또는 MNO(Mobile Network Operator)에 연결되고 동적으로 IP 주소가 할당된다.

이 시점에 인터넷 기반 클라이언트와 서버는 특정 차량에 할당된 동적 IP 주소를 알지 못한다. 따라서 일반적으로 차량은 일반적으로 URL을 사용하여 V2X 서버에 연결하는 잘 알려진 엔드 포인트(endpoint)에 연결해야 한다. 차량과 인터넷 서버는 일반적으로 상호인증을 하고 차량은 암호화된 채널을 통해 VIN(Vehicle Identification Number)과 같은 고유 식별자를 전달하여 V2X 서버에 '등록' 한다. 그 시점부터 V2X 서버는 현재 특정 VIN이 있는 차량에 할당된 IP 주소를 가지며, 이 정보를 다른 인터넷 기반 클라이언트 및 서버와 공유할 수 있고 이 차량에 메시지를 보낼 수 있다.

그러나 차량의 연결이 끊어지고 새 IP 주소가 동적으로 할당되는 경우 새 IP 주소를 다시 등록해야 하며 새 주소를 이해 관계자에게 전달해야 한다. 이는 단순화를 위해 구성 요소 다이어그램에는 표시되지 않는다. 대부분 연결은 차량이 하나 이상의 오프보드 서버(표시됨)에 등록된 후에만 가능하지만, 이러한 시나리오는 본 Vehicle Information Service Specification에서 정의된 인터페이스 및 동작에 영향을 주지 않기 때문이다.

구성 요소 다이어그램은 네 가지 유형의 인터넷 기반 클라이언트를 보여준다. 사용자는 차량의 이더넷 네트워크에 연결된 휴대폰이나 태블릿에서 실행되는 웹 페이지 또는 웹 응용을 사용하여 특정 차량의 차량 신호에 대한 접근을 요청할 수 있다. 또는 이러한 신호는 네이티브 또는 관리형 런타임 애플리케이션을 사용하거나 다른 자동화된 클라이언트 시스템 또는 서비스(예: '스마트 시티'의 트래픽 관리 시스템과 통신하는 온보드 에이전트)를 통해 접근할 수 있다.

⑤ VIS 서버 상태 다이어그램(VIS Server State Diagram)

Figure 3의 다이어그램은 VIS 서버가 가질 수 있는 여러 가능한 상태를 보여준다:

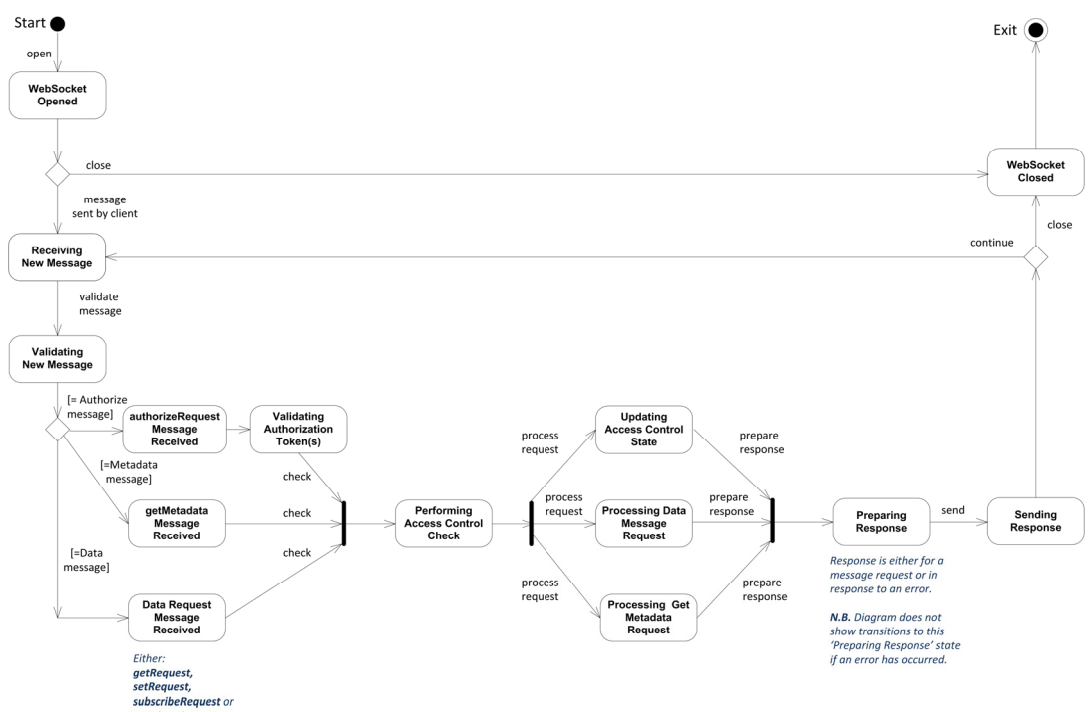


Figure 3. VIS 서버의 상태 다이어그램

상태 다이어그램은 설명용으로만 사용되며 VIS 서버가 단일 또는 다중 스레드라고 가정하지 않는다. VIS 서버의 구현자는 내부 설계 및 구현을 자유롭게 결정할 수 있지만 정상적인 작동에서는 본 사양에서 정의된 API에 따라 서버가 여러 요청을 빠르게 연속적으로 허용할 수 있고 VIS 서버가 각각의 요청을 처리하고 응답할 것이라고 가정한다.

(6) 보안 및 개인 정보 고려 사항(Security and Privacy Considerations)

① 소개

VIS 서버 구현은 하나 또는 그 이상의 차량 신호에 대한 접근을 선택적으로 제한할 수 있어 승인된 사용자 및/또는 디바이스로부터의 요청에 대해서만 응답을 받도록 할 수 있다. 이는 안전, 개인 정보 보호 또는 상업적 고려사항을 비롯한 다양한 이유 때문이다.

따라서 데이터에 대한 GET, SET, SUBSCRIBE 또는 UNSUBSCRIBE 요청은 클라이언트가 요청이 하나 또는 그 이상의 적절히 승인된 보안 주체로부터 온 것임을 서버에 입증해야 한다.

다양한 유형의 보안 원칙이 있다. 신호에 대한 접근 제어를 위해 취한 접근 방식과 데이터 프라이버시의 중요성은 다음 섹션에 설명되어 있다.

② 보안 주체(Security Principals)

보안 주체의 유형은 다음을 포함해야 한다.

타입	설명
사용자 (User)	요청을 담당하는 사람, 시스템 또는 조직(예: 운전자, 응급 서비스, 스마트 시티 교통 관리 시스템)
디바이스 (Device)	요청이 발생한 차량 또는 기기. 예를 들어 차량의 WiFi 핫스팟에 연결된 사용자의 CE(Consumer Electronics) 장치 또는 호송대의 다른 차량이 될 수 있다. 동일한 차량의 ECU(Electronic Control Unit) 또는 인터넷에 연결된 시스템(예: 사물웹(WoT) 장치)

③ 접근 제어 및 승인(Access Control and Authorization)

클라이언트가 신호 데이터에 접근을 요청하면, 예를 들어 특정 사용자 및/또는 차량/디바이스에 대해 하나 이상의 보안 주체를 대신하여 요청을 수행한다.

신호에 대한 접근은 서버에 의하여 관리 및 제어가 수행된다. 서버는 특정 신호 또는 신호 집합에 대한 접근 제어를 허용하지 않을 수 있고, 다른 신호에 대해서는 다른 접근 제어 방법을 사용하도록 선택할 수 있다.

서버에 의해 인증되어야 하는 각 보안 주체에 대해 클라이언트는 보안 토큰(예: OAuth 2.0 토큰(RFC6749 참조))을 획득하여 authorize 액션(action)을 포함한 메시지를 서버에 전달해야 한다.

서버 구현은 특정 신호에 대한 요청이 사용자(예: 차량 운전자)와 클라이언트를 호스팅하는 디바이스(예: 차량) 모두에 대한 보안 토큰을 포함하도록 요구할 수 있다. 다른 차량 신호 세트에 대한 요청은 사용자에게만 권한이 부여되거나 디바이스만 특정 신호에 액세스 할 수 있는 권한을 요구할 수 있다.

클라이언트와 서버는 각각 보안 모범 사례를 구현할 책임이 있다. 여기에는 토큰(해당되는 경우)의 안전한 획득, 그리고 토큰 확인하고, 악의적인 행위자 또는 에이전트에 의한 토큰 재생(replays) 또는 스푸핑(spoofing) 방지가 포함되야 한다. 하지만 이에 국한되지는 않는다.

④ WebSocket 채널 인증(WebSocket Channel Authorization)

클라이언트는 WebSocket 채널의 접근 제어(access-control) 상태를 변경하기 위해 authorize 액션(action)을 포함한 메시지를 사용한다. 메시지 구조는 8장의 메시지 구조에 자세히 정의되어 있다.

Figure 4의 다이어그램은 클라이언트가 차량 속도를 요청하는 시나리오를 보여준다. 이 예제 시나리오에서 신호는 액세스 제어를 받지 않으므로 서버는 요청된 데이터가 포함된 메시지를 반환한다. 그런 다음 클라이언트는 서버에 차량 트렁크 상태를 열림으로 설정하도록 요청하며, 이때 서버는 요청을 처리하기 전에 클라이언트에게 접근 제어 자격 증명 요구한다. 단계 순서는 Figure 4의 다이어그램에 설명되어 있다.

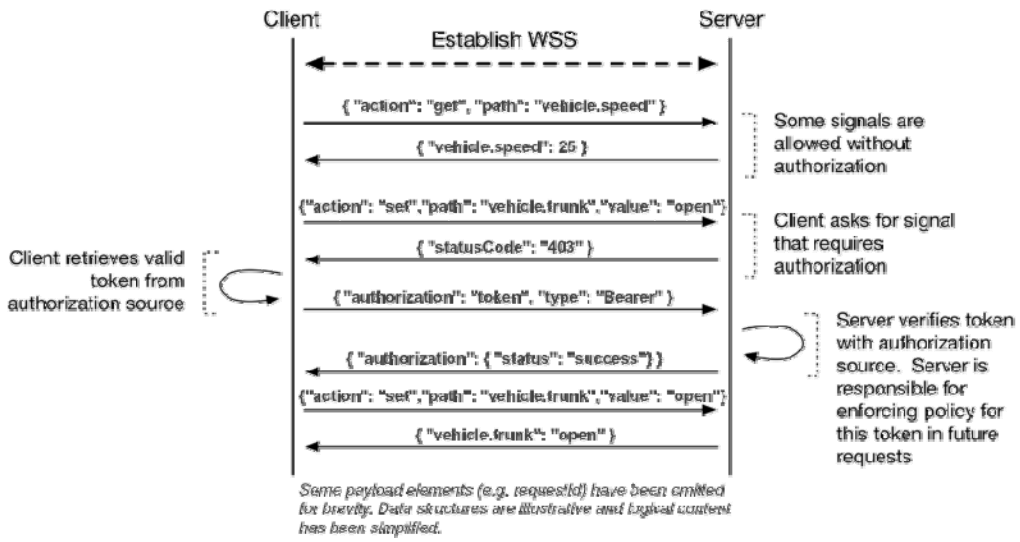


Figure 4. 개념적인 WebSocket 기반 보안 토큰 흐름에 대한 다이어그램

authorize 액션(action)이 포함된 메시지를 받은 후 서버는 예를 들면, 토큰 발급 보안 기관에 확인하여 토큰에 대한 검증을 한다. 서버에 전달된 모든 토큰이 유효한 경우 성공 응답을 반환하고 WebSocket 인스턴스에서 수신한 모든 후속 요청에 상향된 액세스 제어 권한을 갖는다. 특히 각 GET, SET, SUBSCRIBE 및 UNSUBSCRIBE 액션(action)은 서버가 토큰이 대표하는 보안 주체에 적합한 접근 제어 권한을 갖는다.

만약 클라이언트가 다른 토큰 값(들)을 사용하여 후속 authorize 메시지를 서버에 보냈다면, 만일 이들 중 하나 이상이 유효하지 않은 경우 서버는 오류 응답을 반환하고 WebSocket 접근 제어 상태는 변경되지 않는다. 그러나 새 토큰 값이 유효한 경우 서버는 성공 응답을 반환하고 새 토큰과 관련된 접근 제어 권한을 해당 시점 이후의 모든 요청에 적용한다.

클라이언트 또는 서버가 WebSocket 연결을 닫고 새 WebSocket 인스턴스가 열리면 이때는 상향된 접근 제어 권한없이 열린다.

본 사양은 일반적인 사용자 및 디바이스 접근 제어 시나리오를 기반으로 특정 오류 응답을 포함한 접근 제어에 대한 표준화된 토큰기반 접근 방식을 정의한다. 그러나 보안 모델이 확장 가능한 것이 중요하다. 따라서 서버는 본 표준화된 접근 방식과 일치하는 모든 유형의 토큰을 구현할 수 있으며 선택적으로 다른 보안 주체 유형(들)에 대한 추가 토큰을 정의할 수 있다. 이 경우 특정 서버 구현에서 지원하는 보안 토큰의 정확한 유형, 해당 토큰의 형식 및 모든 추가 오류 코드 및 이유(예: 추가 토큰 유형이 만료되어 갱신이 필요함을 표시하는 것)는 서버 문서에 정의되어야 한다.

⑤ 암호화 사용(Use of Encryption)

계층화된 보안 모델을 지원하고 '심층 방어(defence in depth)'를 구축하기 위해 클라이언트와 서버 간의 모든 차량 신호 및 데이터 통신은 강력하게 암호화되어야 한다. 이는 공격자가 요청 데이터 또는 응답 페이로드의 보안 토큰을 도청하거나 변조하는 것을 더 어렵게 만들기 위함이다.

이를 구현할 수 있는 한 가지 방법은 클라이언트와 서버가 공개키 인프라(PKI) 접근 방식을 사용하는 것이다. 여기서 클라이언트는 서버의 X.509 인증서를 확인하여 서버의 ID를 확인하고 클라이언트와 서버는 보안 TLS(Transport Layer Security) '터널' 설정을 협상한다.

WebSocket 프로토콜은 서버가 WebSocket 연결을 오픈하고 클라이언트가 HTTPS를 통해 요청을 수신하도록 요청하면 WebSocket이 TLS를 통해 설정된다. 즉, 안전한 'wss' 연결이 생성된다.

⑥ 토큰 갱신(Token Renewal)

각 보안 토큰은 유효 기간 동안 지정된 수명을 가져야 한다. 접근 제어 대상 신호에 대한 요청을 수신할 때 서버가 토큰이 만료되었기 때문에 요청이 승인되지 않았다고 판단하면 서버는 이러한 이유를 포함하여 오류 응답을 반환한다. 이 경우 클라이언트는 보안 기관(Security Authority)에 새로운 토큰을 요청할 수 있고, 이러한 패턴이 반복된다.

만일 서버가 요청금지로 오류 응답을 반환하는 경우에는 보안 토큰을 갱신해도 요청이 유효하지 않다. 이 경우 클라이언트는 이러한 요청을 반복 수행해서는 안 되며, 다른 변경을 통해 요청을 유효하게 만든 후에 서버에 요청해야 한다.

(7) Websocket 초기화(Initialisation of the WebSocket)

클라이언트 응용이 웹 런타임에서 실행되는 HTML 응용 프로그램이거나 브라우저에서 실행되는 웹 페이지인 경우 WebSocket 인스턴스는 기본적으로 인스턴스화되거나 '표준 준수' WebSocket 자바스크립트(JavaScript) 라이브러리를 사용하여 생성될 수 있다.

WebSocket은 네이티브(예: C++) 응용 또는 Java 또는 C#과 같은 '관리형 런타임' 언어를 사용하여 작성된 응용에서도 실행될 수 있다. 네이티브 및 관리형 클라이언트는 적절한 표준을 준수하는 WebSocket 라이브러리를 사용하여 서버에서 WebSocket 연결에 대한 오픈 요청한다고 가정한다.

추가 디바이스들 또는 여러 VIS 서비스들을 지원하는 구현은 검색 기능을 제공해야 한다. 그렇지

않으면 로컬 차량 네트워크에서 특정 VIS 서버 인스턴스의 위치는 패키지 매니페스트(manifest)의 일부로 구성하거나 응용 프로그램 설치 시 레지스트리를 참조하여 처리할 수 있다. 본 사양에서는 'wwwivi' 호스트 이름이 예로 사용된다.

차량에서 실행되는 클라이언트는 호스트 이름을 사용하여 VIS 서버 인스턴스에 연결할 수 있다. 예를 들어 호스트 이름은 'wwwivi'이며 기본 포트 443을 사용한다. 호스트 이름 'wwwivi'는 로컬 호스트 IP 주소 127.0.0.1에 로컬로 매핑 될 수 있다. 예를 들면 '/ etc / hosts' 파일에 항목을 추가하여 사용하면 된다.

하위 프로토콜 이름은 버전 번호 접미사가 있는 'wvss'이어야 한다. 예를 들면 wvss1.0과 같이 사용할 수 있다. 하위 프로토콜 버전은 정확히 하나의 VSS(Vehicle Server Specification) 버전과 연결되므로 클라이언트와 서버가 요청 및 응답 메시지 패킷을 올바르게 검증하고 구문 분석할 수 있다.

```
var vehicle = new WebSocket("wss://wwwivi", "wvss1.0");
```

클라이언트는 HTTPS를 통해 서버에 연결하고 서버가 WebSocket을 오픈하도록 요청한다. 클라이언트와 서버 간의 모든 WebSocket 통신은 'wss'를 통해 이루어져야 한다. 암호화되지 않은 통신은 지원되지 않으므로 서버는 'ws' 연결 요청은 거부해야 한다.

본 사양은 단일 WebSocket이 클라이언트 응용 프로그램과 서버 간의 통신을 활성화하는 데 사용된다고 가정한다. 클라이언트가 서버에 하나 이상의 WebSocket을 열도록 요청하는 것이 명시적으로 금지되는 것은 아니다. 하지만 서버는 후속 WebSocket 오픈을 거부할 수 있으며 클라이언트는 이를 적절하게 처리할 책임이 있다.

만일 클라이언트 응용과 서버 간에 둘 이상의 WebSocket 연결이 설정되면 각 연결은 독립적으로 관리되어야 한다. 예를 들어 특정 WebSocket 연결을 사용하여 생성된 구독(subscriptions)은 해당 연결을 통해서만 알림을 동작 시켜야 하며 클라이언트는 해당 WebSocket 연결을 사용하여 구독을 취소해야 한다.

만일 하나 이상의 클라이언트와 특정 서버 인스턴스 간에 둘 이상의 WebSocket 연결이 설정된 경우 경합 상태(race conditions) 및 동시성(concurrency) 문제가 발생할 위험이 있다. 이에 대한 예는 두 개 이상의 WebSocket 연결 상황에서 특정 설정을 동시에 업데이트하는 경우이다.

달리 명시적으로 언급하지 않는 한, 클라이언트는 서버가 하나 이상의 WebSocket 연결이 열려

있는 경우 서버는 간단한 동시성 모델을 구현했다고 가정할 수 있으며, 잠재적으로 손실된 업데이트 및 더티 읽기(dirty reads)가 발생할 수 있다

(8) 메시지 구조(Message Structure)

클라이언트는 반드시 WebSocket send 메소드를 사용하여 요청 메시지를 서버에 전달해야 한다. 메시지 서명은 다음과 같아야 한다:

```
WebSocket.send(request)
```

요청 메시지는 이 섹션에 정의된 요청 객체 중 하나로 구성되어야 한다. 클라이언트는 다음과 같이 WebSocket onmessage 메소드를 사용하여 서버로부터 응답을 수신한다.

```
WebSocket.onmessage = function(obj){
                        // process data
                }
```

서버가 반환하는 메시지는 아래 표에 정의된 응답 객체 중 하나여야 한다.

Request Objects	Response Objects
authorizeRequest	authorizeSuccessResponse
	authorizeErrorResponse
metadataRequest	metadataSuccessResponse
	metadataErrorResponse
getRequest	getSuccessResponse
	getErrorResponse
setRequestm	setSuccessResponse
	setErrorResponse
subscribeRequest	subscribeSuccessResponse
	subscribeErrorResponse
	subscriptionNotification
	subscriptionNotificationError
unsubscribeRequest	unsubscribeSuccessResponse
	unsubscribeErrorResponse
unsubscribeAll Request	unsubscribeAllSuccessResponse
	unsubscribeAllErrorResponse

요청 및 응답 매개변수들은 아래 표에 정의된 제한된 수의 애트리뷰트를 포함한다.

용어 정의(Term Definitions)

애트리뷰트	타입	설명
action	Action	클라이언트에서 요청하거나 서버에서 전달되는 action 타입이다
path	String	Vehicle Signal Specification(VSS)에 정의된 것과 같은 원하는 차량 신호에 대한 path이다
requestId	String	클라이언트가 생성한 고유 ID 값이다. 서버가 응답으로 반환하고 클라이언트가 요청 및 응답 메시지를 연결하는 데 사용한다. 값은 정수 또는 UUID(Universally Unique Identifier)일 수 있다
subscriptionId	String	각 구독(subscription)을 고유하게 식별하기 위해 서버에서 반환하는 값이다. 값은 정수 또는 UUID(Universally Unique Identifier)일 수 있다
tokens	object	하나 이상의 보안 토큰(예: OAuth2) 이름/값 쌍을 포함하는 구조(Structure)이다.
timestamp	integer	서버가 응답을 반환한 UTC(협정세계시간) 시간(밀리 초로 표시됨)이다
value	any	서버에서 반환한 데이터 값이다. 이는 기본 타입이거나 JSON 형식의 중첩된 이름/값 쌍으로 구성된 복합 타입일 수 있다.
TTL	integer	인증 토큰의 수명을 초 단위로 반환한다.
filters	object	서버에 구독 요구를 줄이기 위한 필터링 메커니즘을 제공한다.
metadata	object	잠재적으로 사용 가능한 신호 트리를 설명하는 메타데이터이다.
error	Error	오류 코드, 이유 및 메시지를 반환한다.

JSON 스키마 정의

이 섹션의 정의는 JSON Schema VISS 인터페이스 내에서 참조되는 데이터 타입을 설명한다.

```
{
    "definitions": {
        "action": {
            "enum": [ "authorize", "getMetadata", "get", "set", "subscribe",
"subscription", "unsubscribe", "unsubscribeAll"],
            "description": "The type of action requested by the client and/or
```

```
delivered by the server",
        },
        "requestId": {
            "description": "Returned by the server in the response and used by the client to link the request and response messages.",
            "type": "string"
        },
        "path": {
            "description": "The path to the desired vehicle signal(s), as defined by the metadata schema.",
            "type": "string"
        },
        "value": {
            "description": "The data value returned by the server. This could either be a basic type, or a complex type comprised of nested name/value pairs in JSON format.",
            "type": "string"
        },
        "timestamp": {
            "description": "The Coordinated Universal Time (UTC) time that the server returned the response (expressed as number of milliseconds).",
            "type": "integer"
        },
        "filters": {
            "description": "May be specified in order to throttle the demands of subscriptions on the server.",
            "type": ["object", "null"],
            "properties": {
                "interval": {
                    "description": "The server is requested to provide notifications
```

```
with a period equal to this field's value.",
                    "type": "integer"
                },
                "range": {
                    "description": "The server is requested to provide notifications only whilst a value is within a given range.",
                    "type": "object",
                    "properties": {
                      "below": {
                        "description": "The server is requested to provide notifications when the value is less than or equal to this field's value.",
                        "type": "integer"
                      },
                      "above": {
                        "description": "The server is requested to provide notifications when the value is greater than or equal to this field's value.",
                        "type": "integer"
                      }
                    }
                },
                "minChange": {
                    "description": "The subscription will provide notifications when a value has changed by the amount specified in this field.",
                    "type": "integer"
                }
            }
        },
        "subscriptionId":{
            "description": "Integer handle value which is used to uniquely identify the subscription.",
```

```
            "type": "string"
        },
        "metadata":{
            "description": "Metadata describing the potentially available signal tree.",
            "type": "object"
        },
        "error": {
            "description": "Server response for error cases",
            "type": "object",
            "properties": {
                "number": {
                    "description": "HTTP Status Code Number",
                    "type": "integer"
                },
                "reason": {
                    "description": "Pre-defined string value that can be used to distinguish between errors that have the same code",
                    "type": "string"
                },
                "message": {
                    "description": "Message text describing the cause in more detail",
                    "type": "string"
                }
            }
        }
    }
}
```

Action

Action enumeration은 클라이언트가 요청한 작업 타입을 정의하는 데 사용됩니다. 모든 클라이언트 메시지는 action 이름/값 쌍이 있는 JSON 구조를 포함해야 하며, action 프로퍼티(property)의 값은 enumeration에 지정된 값 중 하나여야 한다.

Authorize

클라이언트가 보안 주체에 대한 보안 토큰을 서버에 전달하여 접근 제어를 지원

getMetadata

클라이언트가 잠재적으로 접근할 수 있는 신호 및 데이터 속성을 설명하는 메타데이터를 요청

get

클라이언트가 한번에 하나 또는 하나 이상의 값을 가져올 수 있게 함

set

클라이언트가 한번에 하나 또는 하나 이상의 값을 설정할 수 있게 함

subscribe

클라이언트가 하나 또는 하나 이상의 차량 신호 및/또는 데이터 속성 값을 갖은 JSON 데이터 구조를 포함한 알림 요청. 클라이언트는 서버에서 신호가 변경될 때 알림을 받도록 요청.

subscription

서버가 하나 또는 하나 이상의 차량 신호 및/또는 데이터 속성 값을 갖은 JSON 데이터 구조를 포함한 알림을 클라이언트에 보낼 수 있도록 함

unsubscribe

클라이언트가 해당 구독에 대해 더 이상 알림을 받지 않도록 서버에 요청

unsubscribeAll

클라이언트가 등록된 모든 구독에 대해 더 이상 알림을 받지 않도록 서버에 요청

① Authorize

접근 제어 하에 있는 신호 및 데이터 속성의 접근을 가능하게 하기 위해 클라이언트는 authorize action 메시지를 서버에 선택적으로 전달할 수 있다. 메시지의 구조와 관련 성공 및 오류 응답은 아래에 정의되어 있다.

Authorize Request

authorizeRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
authorizeRequest	action	Action	Yes
	tokens	object	Yes
	requestId	string	Yes


```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Authorize Request",
    "description": "Enables the client to pass security token(s) to the server that can be used to authorize access to signals and data",
    "type": "object",
    "required": ["action", "tokens", "requestId"],
    "properties": {
        "action": {
            "enum": [ "authorize" ],
            "description": "The identifier for the authorize request"
        },
        "tokens": {
            "description": "Extensible key-value pair token mechanism used for access control",
            "type": "object",
            "properties": {
```

```
                "authorization": {
                    "description": "The user token, for the user that the client is making requests on behalf of",
                    "type": "string"
                },
                "www-vehicle-device": {
                    "description": "The device token for the originating device that is making the request to the server",
                    "type": "string"
                }
            }
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

authorizeSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
authorizeSuccessResponse	action	Action	Yes
	TTL	integer	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Authorize Success Response",
```

```
    "description": "The response sent from the server upon a successful authorization request",
    "type": "object",
    "required": ["action", "TTL", "requestId"],
    "properties": {
        "action": {
            "enum": [ "authorize" ],
            "description": "The identifier for the authorize request",
        },
        "TTL": {
            "description": "The time to live of the authorization token",
            "type": "integer"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

authorizeErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
authorizeErrorResponse	action	Action	Yes
	error	Error	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Authorize Error Response",
```

```
    "description": "The response sent from the server upon an unsuccessful authorization request",
    "type": "object",
    "required": ["action", "error", "requestId"],
    "properties": {
        "action": {
            "enum": [ "authorize" ],
            "description": "The identifier for the authorize request",
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

서버는 다음 보안 토큰 타입 및 이름을 지원해야한다:

보안 주체	토큰 이름	설명
User	authorization	클라이언트가 요청을 대신해주는 사용자이다. 사용자는 사람(예: 운전자 또는 승객)일 수 있고, 조직(예: 응급 서비스 기관), 또는 기타 법인이 될 수 있다
Device	www-vehicle-device	서버에 요청하는 발신 디바이스이다. 이것은 VIS 서버를 호스팅하는 차량의 ECU이거나 WiFi 핫스팟을 통해 차량에 연결된 디바이스이거나 다른 디바이스일 수 있다

'토큰(token)' JSON 조각(JSON fragment)은 예를 들어 사용자 토큰만 전달하기 위해 단일 이름/값 쌍만 포함하는 'authorization' 구조를 포함할 수 있다. 또는 차량 토큰만 전달하기 위해 'www-vehicle-device' 이름/값 쌍만 포함할 수 있다. 또는 'authorization' 및 'www-vehicle-device' 토큰 모두에 대한 이름/값 쌍을 포함할 수 있다. 이들은 다음 예에 설명되어 있다.

EXAMPLE 4

```
if(userTokenOnly){
        // Pass user token only
        vehicle.send('{ "action": "authorize",
                "tokens": { "authorization": "<user_token_value>" },
                "requestId": "<some_unique_value>" }');

} else if (deviceTokenOnly) {
        // Pass vehicle/device token only
        vehicle.send('{ "action": "authorize",
                "tokens": { "www-vehicle-device": "<device_token_value>" },
                "requestId": "<some_unique_value>" }');

} else if (userAndDeviceToken) {
        // Pass tokens for user and device
        vehicle.send('{ "action": "authorize",
                "tokens": { "authorization": "<user_token_value>",
                        "www-vehicle-device": "<device_token_value>" },
                "requestId": "<some_unique_value>" }');
}
```

본 사양은 토큰 구조와 토큰을 얻기 위한 방법을 의도적으로 정의하지 않는다. 서버 구현 공급자는 서버에 전달된 토큰의 신뢰성을 확인하기 위해 선호하는 토큰 형식과 방법을 선택할 수 있다. 서버는 토큰을 불투명 구조로 취급하고 평가를 위해 기본 소프트웨어 계층으로 전달할 수도 있다.

클라이언트와 서버는 동일한 토큰 형식을 사용할 것으로 예상된다. 만일 클라이언트가 서버에서 이해하지 못하는 형식을 사용하여 토큰을 제시하면 서버는 토큰을 거부한다.

사전 공격(dictionary attacks)을 더 어렵게 만들기 위해 여러 번의 인증 요청 실패 후 인증이 거부된다. 후속 인증 요청에 대한 응답으로 서버에서는 401(승인되지 않음) too_many_requests 오류를 보낸다. 실패한 요청의 수와 시도가 거부되는 기간은 구현에 의해 결정된다.

② 메타데이타(Metadata)

클라이언트는 잠재적으로 사용가능한 신호 스키마(예: VSS 트리)를 설명하는 메타데이터를 요청하기 위해 getMetadata 액션(action)을 사용할 수 있다. 메타데이터 요청 메시지를 서버에 전송하여 이를 수행한다. 만일 서버가 메타데이터를 반환할 수 있는 경우 metadataSuccess Response 메시지를 사용하여 반환된다. 만일 오류가 발생하면 서버는 metadataErrorResponse 메시지를 클라이언트에 반환한다.

클라이언트는 신호 트리에서 특정 지점부터의 일부분에 대한 메타데이터를 요청할 수 있다. 이런 경우 스키마 계층구조의 지정된 브랜치(branch) 내의 신호에 대한 메타데이터만 반환된다. 예를 들어 섀시(chassis) 경로(path)가 지정되면 VSS 트리의 섀시(chassis) 브랜치에 대한 메타데이터만 반환된다. 경로(path)가 설정되지 않은 경우 응답에 전체 신호 트리에 대한 메타데이터가 포함된다.

동일한 metadataRequest 메시지 내용으로 여러 개의 getMetadata 호출이 다른 시간에 수행되는 경우, metadataSuccessResponse의 메타 데이터는 WebSocket 채널의 액세스 제어 상태가 변경되지 않은 경우 동일해야 한다.

VSS(Vehicle Signal Server) 사양을 사용하면 브랜치(branch)를 public 또는 private으로 정의할 수 있다. VIS 서버는 요청하는 클라이언트와 관련된 접근 제어 권한에 관계없이 public 브랜치의 메타 데이터 요청을 충족한다. 즉, 사용자는 public 브랜치의 특정 신호를 사용할 수 있지만 현재 해당 신호에 대한 Get, Set 또는 Subscribe 권한이 없을 수 있음을 알 수 있다. 브랜치가 'private'로 정의된 경우 적절하게 승인된 클라이언트만 해당 브랜치에 대한 메타데이터를 검색할 수 있다. 이는 차량 제조업체가 상업적으로 민감한 신호 및 데이터에 대한 메타데이터 접근 제어를 적용할 수 있도록 하기 위한 것이다.

클라이언트가 접근할 수 없는 신호를 요청하면 서버는 본 사양의 Get, Set 및 Subscribe 섹션에 설명된 대로 요청을 거부한다. 서버는 비공개로 분류된 브랜치 및 노드에 대해서는 클라이언트가 적절한 접근 권한이 있는 경우에만 메타데이터를 반환해야 한다.

Metadata Request

metadataRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
metadataRequest	action	Action	Yes
	path	string	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Metadata Request",
    "description": "Request metadata describing the potentially available signals",
    "type": "object",
    "required": ["action", "path", "requestId"],
    "properties": {
        "action": {
            "enum": [ "getMetadata" ],
            "description": "The identifier for the getMetadata request"
        },
        "path": {
            "$ref": "#/definitions/path"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

metadataSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
metadataSuccessResponse	action	Action	Yes
	requestId	string	Yes
	metadata	object	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Metadata Success Response",
    "description": "The response sent from the server upon a successful getMetadata request",
    "type": "object",
    "required": ["action", "requestId", "metadata", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "getMetadata" ],
            "description": "The identifier for the getMetadata request"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "metadata": {
            "$ref": "#/definitions/metadata"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

metadataErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
metadataErrorResponse	action	Action	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Metadata Error Response",
    "description": "The response sent from the server upon an unsuccessful getMetadata request",
    "type": "object",
    "required": ["action", "requestId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "getMetadata" ],
            "description": "The identifier for the getMetadata request"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

예제(Examples)

다음 데이터 흐름 예제는 차량 RPM 신호를 포함하는 Signal.Drivetrain. InternalCombustion Engine.RPM 브랜치 내의 신호 구조에 대한 요청을 보여준다. 여기서 메타데이터는 VSS 트리를 사용한다. 이 예제에서는 간결성을 위해 리프 노드(leaf node)를 선택했지만 전체 VSS 브랜치들과 기타 스키마도 getMetadata 인터페이스를 사용하여 요청할 수 있다.

```
client -> {
    "action": "getMetadata",
    "path": "Signal.Drivetrain.InternalCombustionEngine.RPM",
    "requestId": "3874"
  }
  receive <- {
    "action": "getMetadata",
    "requestId": "3874",
    "metadata": { "Signal": {
        "description": "All signals that can dynamically be updated by the vehicle",
        "type": "branch",
        "children": {
          "Drivetrain": {
            "description": "Drivetrain data for internal combustion engines, transmissions, electric motors, etc.",
            "type": "branch",
            "children": {
              "InternalCombustionEngine": {
                "description": "Engine-specific data, stopping at the bell housing.",
                "type": "branch",
                "children": {
                  "RPM": {
                    "description": "Engine speed measured as rotations per minute.",
```

```
                    "min": 0,
                    "max": 20000,
                    "type": "UInt16",
                    "id": 54,
                    "unit": "rpm"
                  }
                }
              }
            }
          }
        }
      },
      "timestamp": 1496087968995
    }
```

다음 데이터 흐름(flow example) 예제는 VSS의 Attribute.Body 브랜치 내의 VSS 구조에 대한 요청을 보여준다. 이 예제에서는 VSS가 Attribute.Body 분기 내에 두 개의 리프 노드(leaf nodes)를 포함한다고 가정한다.

```
client -> {
    "action": "getMetadata",
    "path": "Attribute.Body",
    "requestId": "3875"
  }
  receive <- {
    "action": "getMetadata",
    "requestId": "3875",
    "metadata": { "Attribute": {
```

```
        "description": "Attribute signals that do not change during the power cycle of a vehicle.",
        "type": "branch",
        "children": {
          "Body": {
            "description": "All body components",
            "type": "branch",
            "children": {
              "BodyType": {
                "description": "Body type code as defined by ISO 3779",
                "type": "string"
              },
              "RefuelPosition": {
                "description": "Location of the fuel cap or charge port",
                "type": "string",
                "enum": ["front_left", "front_right", "middle_left", "middle_right", "rear_left", "rear_right"]
              }
            }
          }
        }
      },
      "timestamp": 1489985044000
    }
```


③ Get

클라이언트는 하나 또는 그 이상의 차량 신호 및 데이터 속성의 값을 얻기 위해 서버에 getRequest 메시지를 보낼 수 있다. 만일 서버가 이 요청을 만족시킬 수 있다면 getSuccessResponse 메시지를 반환한다. 만일 서버가 이 요청을 처리할 수 없는 경우(예: 클라이언트가 하나 또는 그 이상의

신호를 검색할 권한이 없기 때문에) 서버는 getErrorResponse를 반환해야 한다. 이러한 메시지 개체의 구조는 다음과 같이 정의된다.

Get Request

getRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
getRequest	action	Action	Yes
	path	string	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Get Request",
    "description": "Get the value of one or more vehicle signals and data attributes",
    "type": "object",
    "required": ["action", "path", "requestId"],
    "properties": {
        "action": {
            "enum": [ "get" ],
            "description": "The identifier for the get request",
        },
        "path": {
            "$ref": "#/definitions/path"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

getSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
getSuccessResponse	action	Action	Yes
	requestId	string	Yes
	value	object	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Get Success Response",
    "description": "The response sent from the server upon a successful get request",
    "type": "object",
    "required": ["action", "requestId", "value", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "get" ],
            "description": "The identifier for the get request",
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "value": {
            "$ref": "#/definitions/value"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

getErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
getErrorResponse	action	Action	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Get Error Response",
    "description": "The response sent from the server upon an unsuccessful get request",
    "type": "object",
    "required": ["action", "requestId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "get" ],
            "description": "The identifier for the get request",
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

예제(Examples)

경로(paths)를 포함하는 모든 예는 설명을 위한 것이라는 점에 유의해야 한다. 유효한 경로 값과 특정 경로에 해당하는 신호 및 데이터 속성은 Vehicle Signal Specification에 정의되어 있다.

아래의 예는 엔진 RPM 값을 얻기 위해 클라이언트가 보낸 getRequest 메시지와 서버가 반환한 getSuccessResponse에 대한 JSON 구조를 보여준다.

```
client -> {
        "action": "get",
        "path": "Signal.Drivetrain.InternalCombustionEngine.RPM",
        "requestId": "8756"
}

receive <- {
        "action": "get",
        "requestId": "8756",
        "value": 2372,
        "timestamp": 1489985044000
}
```

서버가 complex type의 값을 반환해야 하는 경우(예: 즉 단일 기본 JavaScript 유형(예: String, Number, Boolean)이 아닌 값을 반환하는 경우)는 JSON 객체 구조로 이름/값 쌍의 집합으로 반환된다. 이와 같은 형식은 WebSocket이 생성될 때 지정된 WebSocket 하위 프로토콜 값과 관련된 Vehicle Signal Specification의 버전에 의해 정의되어야 한다.

다음은 서버가 complex type에 대한 값으로 getSuccessResponse를 리턴하는 getRequest의 예를 보여준다.

```
client -> {
        "action": "get",
        "path": "Signal.Body.Trunk",
        "requestId": "9078"
}

receive <- {
        "action": "get",
        "requestId": "9078",
        "value": { "Signal.Body.Trunk.IsLocked": false,
                "Signal.Body.Trunk.IsOpen": true },
        "timestamp": 1489985044000
}
```

하나 또는 그 이상의 와일드 카드(별표 '*'로 표시)는 경로(path)의 모든 레벨에 포함되어 해당 레벨의 모든 노드가 포함되도록 지정할 수 있다.

아래 예에서 getRequest의 경로는 모든 도어(door)에 대해 'IsLocked' 상태에 있는 것들만 대상으로 요청하기 위해 리프 노드(leaf node) 위의 레벨에 와일드 카드를 사용했다.

```
client -> {
        "action": "get",
        "path": "Signal.Cabin.Door.*.IsLocked",
        "requestId": "4523"
}

receive <- {
        "action": "get",
        "requestId": "4523",
        "value": [ {"Signal.Cabin.Door.Row1.Right.IsLocked" : true },
```

```
                        {"Signal.Cabin.Door.Row1.Left.IsLocked" : true },
                            {"Signal.Cabin.Door.Row2.Right.IsLocked" : false },
                            {"Signal.Cabin.Door.Row2.Left.IsLocked" : true } ],
            "timestamp": 1489985044000
        }
```

이 예에서는 Signal.Cabin.Door. *"에 대한 응답으로 중첩된 배열을 갖는 complex type이 반환된다. 본 경로(path)는 '모든 문에 대한 모든 신호 및 데이터 속성 반환'을 의미한다. 간단히 표현하기 위해서 이 예제에서는 각 도어(door)는 VSS 정의에 'IsLocked' 및 'Window.Position' 속성 두 개만 정의하고 있다고 가정한다.

```
        client -> {
                "action": "get",
                "path": "Signal.Cabin.Door.*",
                "requestId": "6745"
        }

        receive <- {
                "action": "get",
                "requestId": "6745",
                "value": [ {"Signal.Cabin.Door.Row1.Right.IsLocked" : true,
"Signal.Cabin.Door.Row1.Right.Window.Position": 50},
                                {"Signal.Cabin.Door.Row1.Left.IsLocked" : true,
"Signal.Cabin.Door.Row1.Left.Window.Position": 23},
                                {"Signal.Cabin.Door.Row2.Right.IsLocked" : false,
"Signal.Cabin.Door.Row2.Right.Window.Position": 100 },
                                {"Signal.Cabin.Door.Row2.Left.IsLocked": true,
"Signal.Cabin.Door.Row2.Left.Window.Position": 0 } ],
                "timestamp": 1489985044000
        }
```

다음은 존재하지 않는 데이터 요청 시의 에러 메시지 반환을 보여준다.

```
client -> {
        "action": "get",
        "path": "Body.Flux.Capacitor",
        "requestId": "1245"
}

receive <- {
        "action": "get",
        "requestId": "1245",
        "error": { "number":404,
                "reason": "invalid_path",
                "message": "The specified data path does not exist." },
        "timestamp": 1489985044000
}
```


④ Set

클라이언트는 setRequest 메시지를 서버에 전송하여 서버가 하나 또는 그 이상의 신호 값(예: 하나 이상의 도어를 잠 그거나 창을 연다)을 설정하도록 요청할 수 있다. 만일 서버가 요청을 만족시킬 수 있다면 setSuccessResponse 메시지를 반환한다. 만일 오류가 발생하는 경우(예: 클라이언트가 요청된 값을 설정할 권한이 없거나 값이 읽기 전용인 경우)라면 서버는 setError Response 메시지를 반환한다.

Set Request

setRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
setRequest	action	Action	Yes
	path	string	Yes
	value	any	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Set Request",
    "description": "Enables the client to set one or more values once.",
    "type": "object",
    "required": ["action", "path", "value", "requestId"],
    "properties": {
        "action": {
            "enum": [ "set" ],
            "description": "The identifier for the set request",
        },
        "path": {
            "$ref": "#/definitions/path"
        },
        "value": {
            "$ref": "#/definitions/value"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

setSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
setSuccessResponse	action	Action	Yes
	requestId	string	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Set Success Response",
    "description": "The response sent from the server upon a successful set request",
    "type": "object",
    "required": ["action", "requestId", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "set" ],
            "description": "The identifier for the set request",
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

setErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
setErrorResponse	action	Action	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Set Error Response",
    "description": "The response sent from the server upon an unsuccessful set request",
    "type": "object",
    "required": ["action", "requestId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "set" ],
            "description": "The identifier for the set request",
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

예제(Examples)

신호를 성공적으로 설정한 예제이다.

```
client -> {
        "action": "set",
        "path": "Signal.Cabin.Door.*.IsLocked",
        "value": [ {"Row1.Right.IsLocked": true },
                   {"Row1.Left.IsLocked": true },
                   {"Row2.Right.IsLocked": true },
                   {"Row2.Left.IsLocked": true } ],
        "requestId": "5689"
    }

    receive <- {
        "action": "set",
        "requestId": "5689",
        "timestamp": 1489985044000
    }
```

값의 설정을 실패한 예제로 지정한 경로의 노드가 read-only인 경우이다.

```
client -> {
        "action": "set",
        "path": "Signal.Drivetrain.InternalCombustionEngine.RPM",
        "value": 2000,
        "requestId": "8912"
    }

    receive <- {
```

```
        "action": "set",
        "requestId": "8912",
        "error": { "number": 401,
        "reason": "read_only",
        "message": "The desired signal cannot be set since it is a read only signal"},
        "timestamp": 1489985044000
    }
```

값의 설정을 실패한 예제로 지정된 경로에 값이 없어 오류가 발생한 경우이다.

```
client -> {
    "action": "set",
    "path": "Signal.Drivetrain.InternalCombustionEngine.RPM",
    "value": { "locked" : true }
    "requestId": "2311"
    }

    receive <- {
        "action": "set",
        "requestId": "2311",
        "error": { "number": 400,
        "reason": "bad_request" ,
        "message": "The server is unable to fulfil the client
                    request because the request is malformed."},
        "timestamp": 1489985044000
     }
```

⑤ Subscribe

차량 데이터에 대한 구독(subscriptions)은 서버측 필터링(Server Side Filtering)을 사용하여 달리 지정이 없으면 서버에서 신호가 변경될 때마다 클라이언트에 데이터를 제공한다. 서버는 특히 클라이언트가 지속적으로 변화하는 신호에 가입한 경우, 서버 측의 처리 요구를 줄이기 위해 클라이언트에 전송되는 알림 수를 줄일 수 있다.

클라이언트가 서버에 새 구독 생성을 요청하면 JSON 데이터 개체가 반환된다. 이 오브젝트는 구독을 만들기 위해 서버에 전달된 애트리뷰트와 구독을 고유하게 식별하는 데 사용되는 subscriptionId 정수 핸들 값을 포함한다.

Subscribe Request

subscribeRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
subscribeRequest	action	Action	Yes
	path	string	Yes
	filters	object	No
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Subscribe Request",
    "description": "Allows the client to subscribe to time-varying signal notifications
on the server.",
    "type": "object",
    "required": ["action", "path", "requestId"],
    "properties": {
        "action": {
            "enum": [ "subscribe" ],
            "description": "The identifier for the subscription request"
        },
```

```
            "path": {
                "$ref": "#/definitions/path"
            },
            "filters": {
                "$ref": "#/definitions/filters"
            },
            "requestId": {
                "$ref": "#/definitions/requestId"
            }
        }
}
```

subscribeSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
subscribeSuccessResponse	action	Action	Yes
	requestId	string	Yes
	subscriptionId	string	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Subscribe Success Response",
    "description": "The response sent from the server upon a successful subscription request",
    "type": "object",
    "required": ["action", "requestId", "subscriptionId", "timestamp"],
    "properties": {
```

```
            "action": {
                "enum": [ "subscribe" ],
                "description": "The identifier for the subscription request",
            },
            "requestId": {
                "$ref": "#/definitions/requestId"
            },
            "subscriptionId": {
                "$ref": "#/definitions/subscriptionId"
            },
            "timestamp": {
                "$ref": "#/definitions/timestamp"
            }
        }
}
```


subscribeErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
subscribeErrorResponse	action	Action	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Subscribe Error Response",
    "description": "The response sent from the server upon an unsuccessful subscribe request",
```

```
    "type": "object",
    "required": ["action", "requestId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "subscribe" ],
            "description": "The identifier for the subscription request",
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

subscriptionNotification는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
subscriptionNotification	action	Action	Yes
	subscriptionId	string	Yes
	value	any	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Subscription Notification",
    "description": "Notification sent from the server to provide the requested data
to the client",
    "type": "object",
    "required": ["action", "subscriptionId", "value", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "subscription" ],
            "description": "The identifier for the subscription notification",
        },
        "subscriptionId": {
            "$ref": "#/definitions/subscriptionId"
        },
        "value": {
            "$ref": "#/definitions/value"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

subscriptionNotificationError는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
subscriptionNotificationError	action	Action	Yes
	subscriptionId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Subscription Notification Error",
    "description": "Error message sent by the server when there is an error with an existing subscription",
    "type": "object",
    "required": ["action", "subscriptionId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "subscription" ],
            "description": "The identifier for the subscription notification",
        },
        "subscriptionId": {
            "$ref": "#/definitions/subscriptionId"
        },
        "error": {
            "$ref": "#/definitions/error"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

예제(Examples)

예를 들어 클라이언트가 서버로부터 “Signal.Drivetrain.Transmission.TripMeter” 정보를 수신하기 위해 구독하려는 경우, 다음과 같이 설정된 대상 경로 프로퍼티(property)과 함께 subscribe 액션(action)을 사용할 수 있다.

```
client -> {
        "action": "subscribe",
        "path": "Signal.Drivetrain.Transmission.TripMeter",
        "requestId": "1004"
}

receive <- {
      "action": "subscribe",
      "requestId": "1004",
      "subscriptionId": "35472",
      "timestamp": 1489985044000
}
```

본 예제는 설명 목적으로만 제공되는 것으로 VIS 서버에서 구현한 Vehicle Server Specification 버전에 따라서 ‘Signal.Drivetrain.Transmission.TripMeter’ 경로가 포함되어 있을 수도 있고 그렇지 않을 수도 있다.

만일 subscribeRequest가 성공하면 서버는 위의 예와 같이 subscribeSuccessResponse를 반환하고 클라이언트는 다음 JSON 구조에 기술된 것과 같은 구독 알림을 받기 시작한다.

```
receive <- {
        "action": "subscription",
        "subscriptionId": "35472",
        "value": 36912,
        "timestamp": 1489985044000
}
```

subscriptionId 값은 고유한 값으로, 서버에서 생성되며 해당 WebSocket 인스턴스에 대한 구독을 관리하기 위해 서버 내부적으로 사용한다.

subscriptionId 값은 unsubscribe 액션(action)을 통해 서버에 값을 전달하여 클라이언트가 향후 알림 수신을 구독 해제하는 데 사용한다.

구독 응답을 'GET' 요청에 대한 응답과 구별하기 위해 구독 응답은 해당 알림을 트리거 한 구독을 식별하는 subscriptionId 값을 추가로 포함해야 한다.

서버는 특정 WebSocket 연결에서 성공적인 각 구독 요청에 대해 새로운 고유 subscriptionId 값을 반환한다. 그러나 서버는 구독 핸들 값이 서로 다른 WebSocket 인스턴스 간에 유일성을 보장하지는 않는다.

구독이 서버에 성공적으로 등록되면 클라이언트는 요청된 데이터가 포함된 구독 알림을 받을 수 있다. 알림의 빈도는 서버 측 필터를 사용하여 지정할 수 있다. 기존 구독에 오류가 있는 경우 subscriptionNotificationError가 클라이언트로 전송된다. 이를 통해 클라이언트는 오류 처리(예: 요청된 알림 빈도를 줄이기 위해 필터 조건을 수정)를 할 수 있다.

만일 구독이 활성화 상태인 동안에 인증 토큰이 만료되면 서버는 더 이상 실행할 수 없는 각 구독에 대해 subscriptionNotificationError를 보낸다. 이때 클라이언트는 인증 토큰을 갱신하기 위해 관련 보안 기관과 통신할 책임이 있다.

클라이언트 인증 토큰이 만료된 후에, 서버는 접근 제어 제한이 적용되지 않는 데이터에 대해서는 계속해서 알릴 수 있다. 클라이언트는 갱신된 인증 토큰을 서버로 보낼 때까지는 유효한 인증 토큰이 필요한 구독에 대해서는 알림을 받지 못한다.

만일 새 인증 토큰이 클라이언트에서 적시에 서버에 제공되면 보안 주체가 변경되지 않은 경우 기존의 모든 구독에 대해 서버는 계속 알림을 전송한다. 클라이언트가 재인증하고 구독이 계속되는 기간은 구현에 따라 다르다. 만일 서버가 어떤 구독 처리를 종료한 경우 이를 갱신하는 것은 클라이언트의 책임이다.

서버는 항상 클라이언트가 접근할 수 있는 데이터에 대해서 반드시 구독 알림만 보내야한다.

서버는 WebSocket 연결을 언제든지 닫을 수 있으며 이렇게 함으로써 클라이언트에 대한 모든 구독을 종료할 수 있다. 이때 구독 갱신에 대한 부분은 클라이언트가 담당한다. 클라이언트는 서버 데이터가 더 이상 필요하지 않은 경우 WebSocket 연결을 반드시 닫아야 한다.

구독의 예는 소개(Introduction) 섹션의 예제 1에서 찾을 수 있다.

⑥ Unsubscribe

구독을 취소하려면 클라이언트는 unsubscribeRequest 메시지를 서버에 보낸다. 이는 unsubscribe로 설정된 action 프로퍼티(property)와 subscriptionId에 대한 문자열 값을 포함하는 JSON 구조로 구성된다. 만일 서버가 요청을 충족할 수 있다면 unsubscribeSuccess Response를 반환한다. 예를 들어 잘못된 subscriptionId가 서버로 전달되어 오류가 발생하면 unsubscribeErrorResponse가 반환된다.

만일 클라이언트가 두 개 이상의 WebSocket 인스턴스를 만든 경우 항상 구독 요청 시 원래 사용한 동일한 WebSocket 인스턴스에 대해서 구독을 취소해야 한다. 구독한 신호의 하위 일부부에 대해 구독을 취소할 수 없다. 클라이언트는 원하는 신호에 대한 알림을 수신을 위해 구독을 취소하고 새 구독을 설정해야 한다. 예를 들어 만일 Signal.Body.Lights.* 경로를 사용하여 모든 light에 대한 활성 구독이 있었고 클라이언트는 단지 Signal. Body.Lights.IsLowBeamOn에 대한 정보만 필요하다면, 클라이언트는 Signal.Body.Lights.* 구독을 취소하고 Signal.Body. Lights.IsLowBeamOn에 대해 다시 구독해야 한다.

클라이언트가 더 이상 데이터를 사용하지 않을 때 항상 알림 수신 취소해야 한다. 긴 차량의 여정에서 이는 서버의 처리 부하를 크게 줄이고 서버가 메모리를 확보할 수 있도록 한다. 따라서 서버가 클라이언트의 향후 요청에 지속적으로 응답할 가능성이 높아진다.

Unsubscribe Request

unsubscribeRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeRequest	action	Action	Yes
	subscriptionId	string	Yes
	requestId	string	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Unsubscribe Request",
    "description": "Allows the client to unsubscribe to time-varying signal notifications on the server.",
    "type": "object",
    "required": ["action", "subscriptionId", "requestId"],
    "properties": {
        "action": {
            "enum": [ "unsubscribe" ],
            "description": "The identifier for the unsubscribe request"
        },
        "subscriptionId": {
            "$ref": "#/definitions/subscriptionId"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        }
    }
}
```

unsubscribeSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeSuccessResponse	action	Action	Yes
	subscriptionId	string	Yes
	requestId	string	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Unsubscribe Success Response",
    "description": "The response sent from the server upon a successful unsubscribe request",
    "type": "object",
    "required": ["action", "subscriptionId", "requestId", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "unsubscribe" ],
            "description": "The identifier for the unsubscribe request"
        },
        "subscriptionId": {
            "$ref": "#/definitions/subscriptionId"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

unsubscribeErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeErrorResponse	action	Action	Yes
	subscriptionId	string	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Unsubscribe Error Response",
    "description": "The response sent from the server upon an unsuccessful unsubscribe request",
    "type": "object",
    "required": ["action", "subscriptionId", "requestId", "error", "timestamp"],
    "properties": {
        "action": {
            "enum": [ "unsubscribe" ],
            "description": "The identifier for the subscription request"
        },
        "subscriptionId": {
            "$ref": "#/definitions/subscriptionId"
        },
        "requestId": {
            "$ref": "#/definitions/requestId"
        },
        "error": {
            "$ref": "#/definitions/error"
```

```
        },
        "timestamp": {
            "$ref": "#/definitions/timestamp"
        }
    }
}
```

예제(Examples)

아래 예는 단일 구독에서 구독을 취소할 때 사용되는 unsubscribeRequest 및 unsubscribe SuccessResponse에 대한 JSON 메시지 구조를 보여준다.

```
client -> {
        "action": "unsubscribe",
        "subscriptionId": "102",
        "requestId": "5264"
}

receive <- {
        "action": "unsubscribe",
        "subscriptionId": "102",
        "requestId": "5264",
        "timestamp": 1489985044000
}
```

다음은 잘못된 subscriptionId가 있는 오류 사례의 예이다.

```
client -> {
        "action": "unsubscribe",
        "subscriptionId": "3542",
        "requestId": "7846"
}
receive <- {
        "action": "unsubscribe",
        "subscriptionId": "3542",
        "requestId": "7846",
        "error": { "number":404,
                "reason": "invalid_subscriptionId",
                "message": "The specified subscription was not found." },
        "timestamp": 1489985044000
}
```

EXAMPLE 5

```
// send unsubscribe message
vehicle.send('{ "action": "unsubscribe", "subscriptionId": "102", "requestId": "5429" }');

// set handler
vehicle.onmessage(function(event){
        var msg = JSON.parse(event.data);
        // success case
        if(msg.hasOwnProperty("requestId") && msg.requestId == "5429"){
                console.log("Successfully unsubscribed for id " + msg.subscriptionId);
        // error case
        } else if (msg.hasOwnProperty("error")) {
                console.log("Unsuccessful unsubscribe. " + msg.error.message)
        }
});
```

⑦ Unsubscribe All

모든 구독에서 구독을 취소하려면 클라이언트는 unsubscribeAllRequest 메시지를 서버에 보낸다. 이는 unsubscribeAll로 설정된 액션(action) 프로퍼티(property)을 포함하는 JSON 구조로 구성된다. 이는 subscriptionId 값이 필요하지 않다. 만일 요청이 성공하거나 취소할 활성 구독이 없는 경우 VIS 서버는 unsubscribeSuccessResponse를 반환한다.

Unsubscribe All Request

unsubscribeAllRequest는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeAllRequest	action	Action	Yes
	requestId	string	Yes

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "unsubscribeAll Request",
  "description": "Allows the client to unsubscribe from all notifications on the server.",
  "type": "object",
  "required": ["action", "requestId"],
  "properties": {
    "action": {
      "enum": [ "unsubscribeAll" ],
      "description": "The identifier for the unsubscribeAll request"
    },
    "requestId": {
      "$ref": "#/definitions/requestId"
    }
  }
}
```

unsubscribeAllSuccessResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeAllSuccessResponse	action	Action	Yes
	requestId	string	Yes
	timestamp	integer	Yes

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "unsubscribeAll Success Response",
  "description": "The response sent from the server upon a successful unsubscribeAll request",
  "type": "object",
  "required": ["action", "requestId", "timestamp"],
  "properties": {
    "action": {
      "enum": [ "unsubscribeAll" ],
      "description": "The identifier for the unsubscribeAll request"
    },
    "requestId": {
      "$ref": "#/definitions/requestId"
    },
    "timestamp": {
      "$ref": "#/definitions/timestamp"
    }
  }
}
```

unsubscribeAllErrorResponse는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
unsubscribeAllErrorResponse	action	Action	Yes
	requestId	string	Yes
	error	Error	Yes
	timestamp	integer	Yes

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "UnsubscribeAll Error Response",
  "description": "The response sent from the server upon an unsuccessful unsubscribeAll request",
  "type": "object",
  "required": ["action", "requestId", "error", "timestamp"],
  "properties": {
    "action": {
    "enum": [ "unsubscribeAll" ],
    "description": "The identifier for the unsubscribeAll request"
    },
    "requestId": {
      "$ref": "#/definitions/requestId"
    },
    "error": {
      "$ref": "#/definitions/error"
    },
    "timestamp": {
      "$ref": "#/definitions/timestamp"
    }
  }
}
```

예제(Examples)

아래의 예는 unsubscribeAll 액션(action)에 대한 unsubscribeAllRequest 및 unsubscribe AllSuccessResponse를 보여준다. 이는 클라이언트가 자신이 생성한 모든 구독에 대해서 구독 취소를 하는 데 사용된다.

```
client -> {
    "action": "unsubscribeAll",
    "requestId": "3468"
}

receive <- {
    "action": "unsubscribeAll",
    "requestId": "3468",
    "timestamp": 1489985044000
}
```

(9) Server Side Filtering

필터에 대한 설정을 통해 서버측 필터링을 사용해서 서버 구독 요구를 제한할 수 있다. 이는 트래픽을 줄일 수 있으며, 개발자가 '429-Too Many Requests' 오류 메시지를 받은 경우 사용할 수 있다.

서버측 필터는 확장 가능하므로 추가 필터링 메커니즘을 설정할 수 있다. 현재 특정 범위의 값, 시간 간격 및 최소 변경에 대한 지원이 포함되어 있다. 이는 subscribeRequest 인터페이스의 필터 옵션을 사용하여 구현할 수 있다.

필터는 전체 브랜치가 아닌 VSS 트리의 리프 노드(leaf nodes)와 같은 기본 타입에 적용할 수 있다. 예를 들어, 필터는 Signal.Drivetrain.InternalCombustionEngine.*에 설정할 수 없다. 그러나 필터를 Signal.Drivetrain.InternalCombustionEngine.RPM에는 설정할 수 있다

구독의 기본 동작은 서버에 대상이 차량 데이터가 변경 시 클라이언트에 알림을 보내도록 요청하는 것이다. 이 동작을 수정하기 위해 클라이언트는 필터링 옵션을 사용한다. 다음 필터 옵션이 지원된다.

○ 간격(Interval) - 서버가 매 'n' 밀리 초마다 알림을 제공하도록 요청한다. 이는 만일 클라이언트가 데이터를 더 자주 필요로 하지 않을 경우 클라이언트가 서버의 로드를 줄일 수 있다
○ 범위(Range) - 값이 주어진 범위에 있는 경우에만 알림을 보낸다.
○ 최소 변경(Minimum change) - 값이 지정된 양만큼 변경된 경우에만 알림을 보낸다.

클라이언트는 필터를 사용하여 서버가 다양한 기준에 따라 알림을 보내도록 요청할 수 있지만 이는 요청일 뿐이며 알림 빈도는 궁극적으로 서버에 의해 결정된다는 점에 유의해야한다.

JSON Schema

```
{
  "filters": {
      "$ref": "#/definitions/filters"
  }
}
```


예제(Examples)

다음 구조는 서버측 필터 설정을 요청하는 subscribeRequest 객체의 예이다.

```
//client requests data every 100ms
{ "action": "subscribe", "path": "〈any_path〉",
        "filters": { "interval": 100 },
        "requestId": "〈some_unique_value〉" }

//client requests data when the value is between 100 and 200 (inclusive)
{ "action": "subscribe", "path": "〈any_path〉",
        "filters": { "range": { "above": 100, "below": 200 } },
        "requestId": "〈some_unique_value〉" }
```

```
//client requests data when the value is below 100 (inclusive)
{ "action": "subscribe", "path": "〈any_path〉",
        "filters": { "range": { "below": 100 } },
        "requestId": "〈some_unique_value〉" }

//client requests data when the value changes by 100 units
{ "action": "subscribe", "path": "〈any_path〉",
        "filters": { "minChange": 100 },
        "requestId": "〈some_unique_value〉" }

//client requests data when the value is above 200 (inclusive)
//and the value changes by at least 20 units
{ "action": "subscribe", "path": "〈any_path〉",
        "filters": { "range": { "below": 200 }, "minChange": 20},
        "requestId": "〈some_unique_value〉" }
```

서버에서의 불필요한 로드 방지를 위해 클라이언트는 필요한 것보다 더 작은 최소 변경량을 지정하지 않아야 한다. 서버는 너무 많은 요청을 처리하고 있어 클라이언트가 요청한 요청을 수행할 수 없는 경우 '429-Too Many Request' 오류 응답을 반환한다.

(10) WebSocket 종료(WebSocket Closure)

WebSocket은 WebSocket 인스턴스에서 클라이언트 또는 서버에 의해 'close()' 메서드를 호출하여 종료된다.

다음 예제는 클라이언트에서 WebSocket의 생애(lifetime)를 보여준다.

EXAMPLE 6

```
// Open the WebSocket
var vehicle  = new WebSocket("wss://localhost:4343", "wvss1.0");

// WebSocket is used to GET, SET, SUBSCRIBE and UNSUBSCRIBE
…

// Close the WebSocket
vehicle.close();
```

VIS 서버는 서버에서 결정한 기간 동안 요청을 받지 못한 경우 WebSocket 연결을 종료할 수 있다. 이러한 것을 정상적으로 처리 복구하고 필요한 경우 새 구독을 요청하는 것은 클라이언트의 책임이다.

다음 섹션에서는 서버에서 지원해야하는 오류 응답에 대해 정의한다.

(11) 오류(Errors)

만일 어떤 클라이언트 요청에 오류가 있는 경우 서버는 오류 번호, 이유 및 메시지로 응답한다.

Error 개체는 다음과 같은 애트리뷰트(attribute)를 포함하고 스키마로 정의된다:

Object Name	Attribute	Type	Required
Error	number	integer	Yes
	reason	string	Yes
	message	string	Yes

```
{
  "error": {
      "$ref": "#/definitions/error"
  }
}
```

예제(Examples)

예를 들어 '401(Unauthorized)'과 같은 일부 오류 코드는 원인이 둘 이상일 수 있다. 반환되는 오류 번호는 HTTP 상태 코드 번호(HTTP Status Code Number)(예: 401)이다. 오류 이유도 반환되며, 여기에는 동일한 코드(예: '401 Unauthorized)'가 있지만 원인이 다른 오류를 구별하는 데 사용할 수 있는 사전에 정의된 문자열 값이 포함된다. 오류 메시지는 원인을 자세히 설명하는 메시지 텍스트를 제공하는 데 사용된다.

```
client -> {
          "action": "subscribe",
         "filters": { "<filter_expression>" },
         "path": "<any_metadata_definition>",
         "requestId": "<some_unique_value>"
       }
       receive on error <- {
            "action": "subscribe",
            "requestId": "<some_unique_value>",
            "error":{
              "number": "<error_num>",
              "reason": "<error_reason>",
              "message": "<error_message>"
            },
            "timestamp": "1489985044000"
         }
```

서버 구현은 최소한 아래 표에 나열된 오류 번호와 이유를 지원해야한다.

Error Number (Code)	Error Reason	Error Message
304 (Not Modified)	not_modified	No changes have been made by the server.
400 (Bad Request)	bad_request	The server is unable to fulfil the client request because the request is malformed.
400 (Bad Request)	filter_invalid	Filter requested on non-primitive type.
401 (Unauthorized)	user_token_expired	User token has expired.
401 (Unauthorized)	user_token_invalid	User token is invalid.
401 (Unauthorized)	user_token_missing	User token is missing.
401 (Unauthorized)	device_token_expired	Device token has expired.
401 (Unauthorized)	device_token_invalid	Device token is invalid.
401 (Unauthorized)	device_token_missing	Device token is missing.
401 (Unauthorized)	too_many_attempts	The client has failed to authenticate too many times.
401 (Unauthorized)	read_only	The desired signal cannot beset since it is a read only signal.
403 (Forbidden)	user_forbidden	The user is not permitted to access the requested resource. Retrying does not help.
403 (Forbidden)	user_unknown	The user is unknown. Retrying does not help.
403 (Forbidden)	device_forbidden	The device is not permitted to access the requested resource. Retrying does not help.
403 (Forbidden)	device_unknown	The device is unknown. Retrying does not help.
404 (Not Found)	invalid_path	The specified data path does not exist
404 (Not Found)	private_path	The specified data path is private and the request is not authorized to access signals on this path.
404 (Not Found)	invalid_subscriptionId	The specified subscription was not found.
406 (Not Acceptable)	not_acceptable	The server is unable to generate content that is acceptable to the client
429 (Too Many Requests)	too_many_requests	The client has sent the server too many requests in a given amount of time.

Error Number (Code)	Error Reason	Error Message
502 (Bad Gateway)	bad_gateway	The server was acting as a gateway or proxy and received an invalid response from an upstream server.
503 (Service Unavailable)	service_unavailable	The server is currently unable to handle the request due to a temporary overload or scheduled maintenance (which may be alleviated after some delay).
504 (Gateway Timeout)	gateway_timeout	The server did not receive a timely response from an upstream server it needed to access in order to complete the request.

서버는 선택적으로 추가 오류 코드를 반환할 수 있다. 이 경우 서버 문서에 정의되어 있을 것으로 예상된다. 가능한 경우 서버는 오류 조건에 대해 정의된 표준 HTTP 오류 코드를 반환한다. 예를 들어 RFC7231, RFC7235 및 RFC6585를 참고할 것을 권고한다.

(12) 참고문헌(References)

① Normative references

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[VSS]

Vehicle Signal Specification. GENIVI Alliance. March 2017. 1.0. URL: https://github.com/GENIVI/vehicle_signal_specification

[WEBSOCKETS-API]

The WebSocket API. Ian Hickson. W3C. 20 September 2012. W3C Candidate Recommendation. URL: https://www.w3.org/TR/websockets/

② Informative references

[WEBSOCKETS-PROTOCOL]

The WebSocket Protocol. I. Fette; A. Melnikov. IETF. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6455

2) Vehicle Information Service Specification 해설

(1) 표준 1장 소개 부분에 대한 해설

먼저 본 표준의 목적 및 활용 관점에서 간단하게 설명하면, VISS는 차량 데이터 포맷에 독립적으로 설계된 차량 정보 접근을 위한 인터페이스 표준이다. 본 표준은 자율주행차에 탑재되는 차량 서버에 구현이 될 것을 가정하고 있다. 즉 차량에서 차량 데이터를 접근할 수 있는 표준 인터페이스를 외부로 제공하고, 차량의 내외부의 응용이나 서비스들이 본 표준 인터페이스를 이용하여 차량 데이터를 접근 및 활용할 수 있다. 기본적으로 차량 제조사에서 자사의 차량 데이터 클라우드를 만들 때 본 인터페이스를 사용하여 자사에서 판매한 차들로부터 차량 데이터를 수집할 수 있을 것이다.

차량 데이터 접근을 위한 인터페이스를 정의할 때 크게 두 가지의 표준이 필요하다. 하나는 차량데이터 접근에 필요한 인터페이스와 프로토콜에 대한 표준이며, 다른 하나는 차량 데이터를 기술하는 포맷에 대한 표준이다. 이들 두 개의 표준은 아래와 같으며 본 문서에서는 VISS 표준을 중심으로 설명한다.

- VISS(Vehicle Information Service Specification): W3C의 차량 정보 접근을 위한 인터페이스 및 프로토콜 표준
- VSS(Vehicle Signal Specificatioin): GENIVI 얼라이언스의 차량 데이터 포맷 표준

본 표준에서는 차량의 데이터를 설명 시 '신호'와 정적 데이터라는 용어가 사용된다. '신호'는 차량 속도와 같이 지속적으로 변하는 차량 데이터를 의미하며, 정적 데이터는 말 그대로 차량이 출시될 때 결정되는 차량의 길이, 무게 등 변하지 않는 데이터를 의미한다.

본 표준에서는 차량 데이터 접근을 위한 프로토콜을 JSON 기반의 메시징 프로토콜로 정의를 하며 이는 기본적으로 WebSocket 통신을 사용한다. HTML5의 양방향 통신 방법인 WebSocket 통신을 기반으로 클라이언트와 서버가 연결을 설정하고, 실제적인 Request/Response는 JSON 기반의 메시지를 사용하여 서로 통신한다. 에러 코드의 경우는 HTTP의 에러코드를 확장하여

정의하여 기존의 웹 개발자들이 쉽게 이해하고 활용할 수 있도록 하였고, 인증 메커니즘은 OAuth를 사용하여 Access Token을 사용한다.

(2) 표준 2장 적합성(Conformance)에 대한 해설

VISS 표준 문서의 내용을 표준(normative) 부분과 비표준(non-normative) 부분으로 나눌 수 있는데 이에 대해 설명한다.

(3) 표준 3장 용어(Terminology)에 대한 해설

VISS, VSS 및 WebSocket에 용어에 대한 설명으로 이는 앞에서 설명하였다.

(4) 표준 4장 그림표(Table of Figures)에 대한 해설

본 표준의 그림에 대한 리스트이다

(5) 표준 아키텍처(Architecture)에 대한 해설

VISS에서 사용하는 차량 데이터 포맷인 VSS는 트리 구조로 되어 있으며, 점 표기법(dot notation)을 사용한다. 4장의 Figure 1은 '차량 신호 트리의 예를 보여주는 다이어그램'을 기준으로 설명한다. 일단 그림의 하단에 signal, branch, attribute 세 가지 블록 그림이 있다. 초록색 바탕의 signal 블록은 지속적으로 변하는 동적 데이터인 '신호' 정보를 의미한다. 반대로 설명하면 신호 정보를 VSS 트리의 다이어그램에서 표시할 때는 초록색 바탕의 블록으로 표시한다. Branch의 경우는 데이터가 아니라 연결해주는 가지 역할을 하는 블록으로 초록색 테두리에 초록색 글씨로 표현한다. 마지막으로 attribute는 값이 변하지 않는 정적 데이터를 의미하며 VSS 다이어그램에서 검은색 테두리에 검은색 글씨로 표현된다.

VISS 표준에서 특정 차량 데이터 정보를 요청할 때 점 표기법(dot notion)으로 표현하며 아래의 예와 같이 표현할 수 있다. 다만 점 표기법은 트리의 모양에 따라서 달라지기 때문에 어떤 버전의 VSS 표준을 사용하는지 또는 VSS 이외의 다른 차량 데이터 모델을 사용하는지에 따라서 변경될 수 있어 실제 구현이 이에 대한 확인이 필요하다.

- 차량의 엔진 rpm 데이터: engine.rpm
- 차량의 무게 데이터: body.weight

○ 차량의 왼쪽 사이드 미러의 열선 상태 데이터: body.mirrors.left.heated

VISS 표준을 지원하는 서버를 VIS 서버로 칭한다. 4장의 Figure 2 '차량 시스템, VIS 서버 및 해당 클라이언트 간의 관계를 보여주는 다이어그램'을 기준으로 전체적이 아키텍처에 대해서 설명한다. 기본적으로 그림의 VIS 서버는 VISS에서 정의한 표준 인터페이스를 지원하며, 본 표준이 웹소켓과 JSON 기반의 프로토콜을 사용하기 때문에 어떤 응용도 VIS 서버와 통신이 가능하다. 이것을 Web App, Web Agent, Native or Managed Agent, Native or Managed App 블록을 만들어 그린 것이다. 다만 Web App이나 Web Agent 같은 경우는 자바스크립트 라이브러리 활용할 수도 있고 그렇지 않고 직접 구현할 수도 있다. Internet이라는 큰 블록의 V2X Servers는 외부에서 차량 데이터를 접근하는 클라우드 기반의 서버들로 가정할 수 있고 이러한 클라우드 차량 데이터 플랫폼에서도 VISS 표준을 외부로 제공하여 다양한 차량 데이터를 활용한 응용이나 서비스 개발을 지원할 수 있다.

(6) 표준 보안 및 개인 정보 고려 사항(Security and Privacy Considerations)에 대한 해설

VIS 서버는 보안과 개인정보보호를 위해 차량 신호 접근에 대한 제한을 할 수 있고, 승인된 사용자 및/또는 디바이스로부터의 요청에 대해서만 응답할 수 있다. 이는 차량 데이터의 특성에 따라서 다른 접근 정책이 적용될 수 있다.

보안 주체를 user, device로 크게 부분하고 있다. user는 요청을 담당하는 사람, 시스템 또는 조직이 될 수 있다. 예를 들어 운전자, 응급 서비스 시스템, 스마트 시티 교통 관리 시스템 등이 될 수 있다. device는 요청이 발생한 차량 또는 디바이스를 의미한다. 예를 들어 차량의 WiFi 핫스팟에 연결된 사용자의 CE(Consumer Electronics) 장치 또는 호송대의 다른 차량이 될 수 있다. 동일한 차량의 ECU(Electronic Control Unit) 또는 인터넷에 연결된 시스템(예: 사물웹(WoT) 장치)도 될 수 있다.

기본적으로 클라이언트에 대한 인증은 OAuth 기반의 보안 토큰을 사용한다. 차량이 데이터에 따라서 다른 보안 레벨을 정의할 수 있다. 이런 경우 보안 레벨이 높은 정보를 접근하기 위해서는 보안 레벨이 높은 토큰을 토큰 발급 서버로부터 받아야 한다.

토큰 기반의 인증을 활용한 차량 데이터 접근에 대한 전체적인 흐름에 대한 예는 Figure 4를 참고하면 된다. 본 그림에서 중요한 것은 데이터 접근에 필요한 적절한 인증 토큰 없이 서버에 데이터를 요청하는 경우 서버는 요청을 거부하며 적절한 토큰을 요구하게 된다는 것이다. 그리고 클라이언트는 이를 이해하고 원하는 데이터 접근에 맞는 토큰을 발급받아 다시 요청해야 한다는 것이다. 이러한 방식이 본 표준에서 사용하는 차량 데이터 접근을 관리하는 핵심적인 방법이다.

토큰 갱신의 경우는 VISS를 적용하는 기업들에 따라서 전략적으로 적절히 정의하여 사용할 수 있다.

VISS 표준은 기본적인 보안을 강화를 위해 기본적으로 TLS 기반의 통신을 한다. 즉, Secure WebSocket 통신인 'wss'를 사용한다. 만일 클라이언트가 TLS를 사용하지 않는 WebSocket 통신인 'ws'으로 VIS 서버에 연결을 요청하면 서버는 연결을 거부한다.

(7) 표준 Websocket 초기화(Initialisation of the WebSocket)에 대한 해설

VISS에서 사용하는 WebSocket은 HTML5의 표준을 그대로 사용하기 때문에 기존에 Web Socket을 활용하여 프로그램을 개발한 경험이 있는 개발자는 매우 쉽게 개발이 가능하다. 다만 VISS 표준의 관점에서는 아래의 두 가지 경우에 대해서는 추가적인 이해가 필요하다.

첫 번째는 WebSocket 초기화에서 반드시 이해가 필요한 부분으로 VISS에서는 하위 프로토콜을 의미있게 사용한다는 것이다. 즉, 하위 프로토콜 이름은 버전 번호 접미사가 있는 'wvss'이어야 한다. 예를 들면 wvss1.0과 같이 사용할 수 있다. 하위 프로토콜 버전은 차량 데이터 포맷에 대한 표준인 VSS(Vehicle Server Specification) 표준 버전과 연계하여 사용한다. VSS 표준도 지속적으로 수정이 되고 있기 때문에 클라이언트가 서버 연결 시 설정한 서브 프로토콜 버전에 따라서 클라이언트와 서버 간의 데이터 교환에 사용한 VSS 표준이 달라진다. 따라서 이 부분을 명확히 이해하지 못하면 클라이언트와 서버 간의 적절한 데이터 교환이 불가능할 수 있다.

두 번째는 클라이언트가 두 개 이상의 WebSocket 인스턴스를 사용하는 경우 섹션의 컨텍스트가 섹션 별로 독립되어 운영된다는 것이다. 즉, 클라이언트가 1번 세션에서 구독을 요청한 경우 이에 대한 응답은 1번 세션으로 온다. 따라서 구독을 해지할 경우도 1번 세션을 통해서 해야 한다는 것이다. 또한 경합 상태(race conditions) 및 동시성(concurrency) 문제가 발생할 위험도 있다는 것을 이해할 필요가 있다.

(8) 표준 메시지 구조(Message Structure)에 대한 해설

본 장이 VISS 표준의 가장 핵심적인 부분이라고 할 수 있다. 이 장에서는 Get, Set, Subscribe, Unsubscribe, UnsubscribeAll에 대한 구체적인 메시지 포맷에 대해서 정의한 내용이다.

일단 '용어 정의(Term Definitions)' 테이블에 대해서 이해가 필요하다. 이 테이블의 애트리뷰트가 각 액션(action)의 메시지를 구성하는 주요 정보이기 때문이다. 또한 JSON 스키마는 각 메시지 포맷에서 사용되는 enumeration type의 값들과 명확한 메시지 포맷을 정의하고 있다.

또한 클라이언트와 서버 간에 보낼 수 있는 액션(action)은 반드시 이해가 필요하며 아래와 같다.

Action

Action enumeration은 클라이언트가 요청한 작업 타입을 정의하는 데 사용된다. 모든 클라이언트 메시지는 action 이름/값 쌍이 있는 JSON 구조를 포함해야 하며, action 프로퍼티(property)의 값은 enumeration에 지정된 값 중 하나여야 한다.

Authorize

클라이언트가 보안 주체에 대한 보안 토큰을 서버에 전달하여 접근 제어를 지원

getMetadata

클라이언트가 잠재적으로 접근할 수 있는 신호 및 데이터 속성을 설명하는 메타데이터를 요청

get

클라이언트가 한 번에 하나 또는 하나 이상의 값을 가져올 수 있게 함

set

클라이언트가 한 번에 하나 또는 하나 이상의 값을 설정할 수 있게 함

subscribe

클라이언트가 하나 또는 하나 이상의 차량 신호 및/또는 데이터 속성 값을 갖은 JSON 데이터 구조를 포함한 알림 요청. 클라이언트는 서버에서 신호가 변경될 때 알림을 받도록 요청

subscription

서버가 하나 또는 하나 이상의 차량 신호 및/또는 데이터 속성 값을 갖은 JSON 데이터 구조를 포함한 알림을 클라이언트에 보낼 수 있도록 함

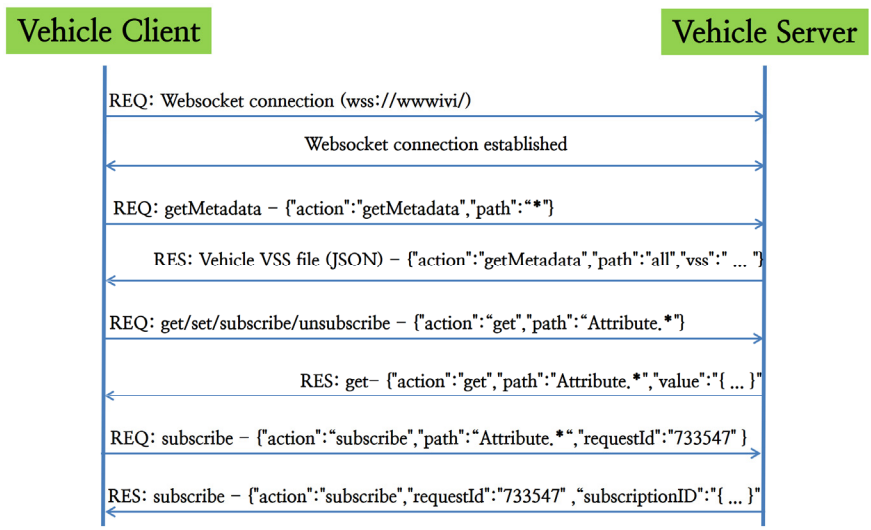
unsubscribe

클라이언트가 해당 구독에 대해 더 이상 알림을 받지 않도록 서버에 요청

unsubscribeAll

클라이언트가 등록된 모든 구독에 대해 더 이상 알림을 받지 않도록 서버에 요청

클라이언트와 서버 간의 전체적인 흐름을 아래 그림을 통해서 설명한다.



클라이언트는 인증서 기반의 웹소켓인 wss을 통해서 VIS 서버에 연결한다. 그리고 'getMetadata' 액션(action)을 활용하여 차량이 제공하는 데이터 세트에 대한 메타데이터를 요청한다. 클라이언트가 서버로부터 메타데이터를 받으면 메타데이터를 분석하여 차량이 제공할 수 있는 데이터에 대해서 이해한다. 그후부터 클라이언트는 'get, set, subscribe, unsubscribe, unsubscribeAll' 등 다양한 action을 활용하여 서버에 데이터를 요청한다. 요청 메시지에 포함되는 "requestId"는 요청과 응답 메시지를 동시에 여러 개씩 주고 받을 때 어떤 요청에 대한 응답 메시지인지를 확인하기 위한 정보이다.

'authorize, getMetadada, get, set, subscribe, subscription, unsubscribe, unsubscribeAll'의 메시지에 대해서 각각의 예제를 보면 상세한 메시지 구성을 이해할 수 있다.

(9) 표준 9장 서버측 필터링(Server Side Filtering)에 대한 해설

VISS 표준의 구독 기능을 사용할 때 서버측 필터링은 서버가 보다 효율적으로 클라이언트로 알림을 제공할 수 있도록 지원하는 기능이다. 클라이언트가 구독을 요청하면 서버는 기본적으로 알림 대상 정보의 변경시에 클라이언트에 알림을 제공하는 것이다. 서버측 필터링은 여기에 알람의 간격 설정 기능, 값의 범위 설정 기능 및 최소 변경 설정 기능을 추가적으로 사용할 수 있도록 하는 것이다.

(10) 표준 10장 WebSocket 종료(WebSocket Closure)에 대한 해설

클라이언트 또는 서버에 의해 모두 WebSocket 연결을 종료할 수 있으며, 비정상적인 종료 등 다양한 상황에서 연결에 대한 책임은 클라이언트에 있다. 즉, 클라이언트 프로그램에서 다양한 상황을 고려하여 목적에 맞게 로직을 개발해야 한다.

(11) 표준 11장 오류(Errors)에 대한 해설

오류 객체는 오류 번호, 이유 및 메시지로 구성되며, 표로 구성된 에러 정보를 참고하면 된다.

(12) 참고문헌(References)에 대한 해설

VSS, WebSocket 등 표준을 참고하면 된다.

1. 본 「W3C 자율주행차 차량정보 접근 인터페이스 표준해설서」는 정부(과학기술정보통신부)의 재원으로, 정보통신기획평가원의 지원을 받은 과제(2017-0-00060, 사실표준화기구 전략대응 및 국제표준화전문가 활동강화)연구결과로 발간된 자료입니다.

◪ 총 괄 책 임 자 : 구경철(TTA 표준화본부장)
◪ 사 업 책 임 자 : 김대중(TTA 표준기획단장)
◪ 작성 및 검수자 : 이원석(ETRI), 인민교(ETRI)
◪ 표 준 기 획 단 : 김영재, 김정헌, 전철기, 진수경, 전지윤, 임영선, 오지훈

웹소켓 통신 기반

W3C 자율주행차 차량정보 접근 인터페이스 표준해설서 (Ver.2020)

2020년도 12월 인쇄
2020년도 12월 발행

발 행 소 : 한국정보통신기술협회
발 행 인 : 최 영 해
발간번호 : TTA-20115-SD
인 쇄 인 : (주)명진씨엔피 (02-2164-3000)

한국정보통신기술협회
Telecommunications Technology Association

463-824, 경기도 성남시 분당구 분당로 47
Tel : 031-780-9178, Fax : 031-724-0109
http://www.tta.or.kr