

Avaya+Oceana+Reference+Frontend+SDK+Guide



Version 17, October, 2021
© 2021 Avaya Inc.
All Rights Reserved.

- Purpose and Intended Audience
- The Web UI
 - 1. Supported Browsers
 - 2. Deploying the Web UI
 - 3. Configuring the Web UI
 - 3.1. Entry points
 - 3.2. Third-Party libraries
 - 3.3. Using the configuration panel
 - 3.4. Secure Connections
 - 3.5. Server-side configuration for Web Chat
 - 4. Upgrading the Web UI
 - 4.1. Upgrading from Oceana 3.2.1 to Oceana 3.2.2.x
 - 4.2. Upgrading from Oceana 3.2.2.1 to Oceana 3.3.0.0
 - 4.3. Upgrading from 3.3.0.0 to 3.4.0.0
 - 4.4. Upgrading from 3.4.0.0 to 3.4.0.1
 - 4.5. Upgrading from 3.4.x to 3.5.0.0
 - 4.6. Upgrading from 3.5.0.0 to 3.5.0.1
 - 4.7. Upgrading from 3.5.x to 3.6.0.0
 - 4.8. Upgrading from 3.6.0.0 to 3.6.1.0
 - 4.9. Upgrading from 3.6.x to 3.7.0.0
 - 4.10. Upgrading from 3.7.0.0 to 3.7.0.1
 - 4.11. Upgrading from 3.7.x to 3.8
 - 4.12. Upgrading from 3.8.0.0 to 3.8.0.1
 - 4.13. Upgrading to 3.8.1
 - 5. Customising the Web UI
 - 5.1. How to edit the UI
 - 5.2. Opening and closing a chat
 - 5.3. Which methods to call for chat
 - 5.4. Configuring the appearance of the chat panel
 - 5.5. Setting custom fields
 - 5.6. Passing FileTransferNotifications through a proxy (Oceana 3.2.2.1 and earlier)
 - 5.7. Editing or suppressing agent presence notifications
 - 5.8. Displaying Co-Browsing URLs in the chat transcript
 - 5.9. Adding or removing log messages
 - 5.10. Adding chat attributes
 - 5.11. Changing the Estimated Wait Time request and response
 - 5.12. Removing the Configuration Panel
 - 5.13. Resetting the chat panel upon completion of a chat
 - 5.14. Localisation
 - 5.15. Redirecting to a survey post chat completion
 - 5.16. Passing FileTransferNotifications through a proxy (pre-Oceana 3.6.1.0)
 - 5.17. Removing inline scripts and CSS
 - 5.18. Setting the authentication token
 - 6. The Customer Journey
 - 6.1. Short sequence (prior to Oceana 3.6.1.0)
 - 6.2. Short sequence (3.6.1.0 and later)
 - 6.3. Initialising
 - 6.4. Generating the Customer ID
 - 6.5. Adding Customer Journey attributes
 - 7. Testing the Web UI
 - 8. Troubleshooting the UI
 - 8.1. General troubleshooting procedure for Web Chat
 - 8.2. Common errors
- The Co-Browse UI
 - 9. Co-Browse Page Dependencies
 - 10. Co-Browse UI Element IDs
 - 11. Co-Browse UI Dialogs
 - 12. Types of Co-Browse Interactions
 - 13. Frequently Asked Questions
- Reference Network Architecture
- Security Considerations
 - 14. Ports

- 15. Secure Transmission
- 16. Installing the Automated Chat Server Certificate
 - 16.1. Obtaining the certificate
 - 16.2. Installation of Automated Chat Server Certificate in System Manager
- 17. Securing Apache proxy servers
- 18. Authentication & Session Management
- 19. Authorisation
- 20. Denial of Service Prevention
- 21. Data Validation
- 22. Data Protection
- 23. Co-Browse
- 24. Defining and implementing a Content-Security-Policy
 - 24.1. Limitations
- The Android Reference App
 - 25. Disclaimer for Android reference App:
 - 26. Introduction
 - 27. Capabilities
 - 28. The archive contains:
 - 29. QuickStart to your first conversation
 - 29.1. Installing Android Studio
 - 29.2. Adding the URLs, Work-Flow and Route-point for the app
 - 29.3. Adding Attributes to the app
 - 29.4. Chat room
 - 30. Customization and implementation details
 - 30.1. EWT
 - 30.2. Websocket handling
 - 30.3. Changing the appearance of the app
 - 30.4. Configuring secure connections
 - 30.5. Upgrading from 3.2.1 to 3.2.2.X
 - 30.6. Permissions
 - 31. Testing the Android App
 - 32. Limitations of the Android App
 - 33. Log locations
- The iOS Reference App
 - 34. Disclaimer for iOS reference App:
 - 35. Introduction
 - 36. Capabilities
 - 37. QuickStart to your first conversation
 - 37.1. Chat room
 - 38. Customising the iOS App
 - 38.1. Configuring secure connections
 - 38.2. EWT
 - 38.3. Websocket handling
 - 38.4. Upgrading from 3.2.1 to 3.2.2.X
 - 38.5. Log locations
 - 39. Testing the iOS App
 - 40. Limitations of iOS App
- Transcript Filtering
 - 41. Enabling transcript filtering
 - 42. Testing
 - 42.1. Filter Request and Response (legacy/basic version)
 - 42.2. Filter request and response (enhanced)
- Web UI API
 - 42.3. Notation
 - 43. WebSocket messages
 - 44. REST requests and responses
 - 44.1. Request Estimated Wait Time
 - 44.2. Request Customer ID
 - 44.3. Create Context Request
 - 44.4. Context ID Response
 - 44.5. Customer Journey Context Request/Response
 - 44.6. Update Context
 - 44.7. Update topic

AVAYA SOFTWARE DEVELOPMENT KIT LICENSE AGREEMENT

REVISED: October 14, 2019

READ THIS CAREFULLY BEFORE ELECTRONICALLY ACCESSING OR USING THIS PROPRIETARY PRODUCT!

THIS IS A LEGAL AGREEMENT ("AGREEMENT") BETWEEN YOU, INDIVIDUALLY, AND/OR THE LEGAL ENTITY FOR WHOM YOU ARE OPENING, INSTALLING, DOWNLOADING, COPYING OR OTHERWISE USING THE AVAYA SOFTWARE DEVELOPMENT KIT ("SDK") (COLLECTIVELY, AS REFERENCED HEREIN, "YOU", "YOUR", OR "LICENSEE") AND AVAYA INC. OR ANY AVAYA AFFILIATE (COLLECTIVELY, "AVAYA"). IF YOU ARE ACCEPTING THE TERMS AND CONDITIONS OF THIS AGREEMENT ON BEHALF OF A LEGAL ENTITY, YOU REPRESENT AND WARRANT THAT YOU HAVE FULL LEGAL AUTHORITY TO ACCEPT ON BEHALF OF AND BIND SUCH LEGAL ENTITY TO THIS AGREEMENT. BY OPENING THE MEDIA CONTAINER, BY INSTALLING, DOWNLOADING, COPYING OR OTHERWISE USING THE AVAYA SOFTWARE DEVELOPMENT KIT ("SDK") OR AUTHORIZING OTHERS TO DO SO, YOU SIGNIFY THAT YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT HAVE SUCH AUTHORITY OR DO NOT WISH TO BE BOUND BY THE TERMS OF THIS AGREEMENT, SELECT THE "DECLINE" BUTTON AT THE END OF THE TERMS OF THIS AGREEMENT OR THE EQUIVALENT OPTION AND YOU SHALL HAVE NO RIGHT TO USE THE SDK.

1.0 DEFINITIONS.

1.1 "Affiliates" means any entity that is directly or indirectly controlling, controlled by, or under common control with Avaya Inc. For purposes of this definition, "control" means the power to direct the management and policies of such party, directly or indirectly, whether through ownership of voting securities, by contract or otherwise; and the terms "controlling" and "controlled" have meanings correlative to the foregoing.

1.2 "Avaya Software Development Kit" or "SDK" means Avaya technology, which may include Software, Client Libraries, Specification Documents, Software libraries, application programming interfaces ("API"), Software tools, Sample Application Code and Documentation.

1.3 "Client Libraries" mean any enabler code specifically designated as such and included in a SDK. Client Libraries may also be referred to as "DLLs", and represent elements of the SDK required at runtime to communicate with Avaya products or other SDK elements.

1.4 "Change In Control" shall be deemed to have occurred if any person, entity or group comes to own or control, directly or indirectly, beneficially or of record, voting securities (or any other form of controlling interest) which represent more than fifty percent (50%) of the total voting power of the Licensee.

1.5 "Derivative Work(s)" means any translation (including translation into other computer languages), port, compiling of Source Code into object code, combination with a pre-existing work, modification, correction, addition, extension, upgrade, improvement, compilation, abridgment or other form in which an existing work may be recast, transformed or adapted or which would otherwise constitute a derivative work under the United States Copyright Act. Permitted Modifications will be considered Derivative Works.

1.6 "Documentation" includes programmer guides, CDs, manuals, materials, and information appropriate or necessary for use in connection with the SDK. Documentation may be provided in machine-readable, electronic or hard copy form.

1.7 "Intellectual Property" means any and all: (i) rights associated with works of authorship throughout the world, including copyrights, neighboring rights, moral rights, and mask works, (ii) trademark and trade name rights and similar rights, (iii) trade secret rights, (iv) patents, algorithms, designs and other industrial property rights, (v) all other intellectual and industrial property rights (of every kind and nature throughout the world and however designated) whether arising by operation of law, contract, license, or otherwise, and (vi) all registrations, initial applications, renewals, extensions, continuations, divisions or reissues thereof now or hereafter in force (including any rights in any of the foregoing).

1.8 "Permitted Modification(s)" means Licensee's modifications of the Sample Application Code as needed to create applications, interfaces, workflows or processes for use with Avaya products.

1.9 "Specification Document" means any notes or similar instructions in hard copy or machine readable form, including any technical, interface and/or interoperability specifications that define the requirements and conditions for connection to and/or interoperability with Avaya products, systems and solutions.

1.10 "Source Code" means human readable or high-level statement version of software written in the source language used by programmers and includes one or more programs. Source Code programs may include one or more files, such as user interface markup language (.mxml), action script (.as), precompiled Flash code (.swc), java script (.js), hypertext markup language (.html), active server pages (.asp), C# or C# .Net source code (.cs), java source code (.java), java server pages (.jsp), java archives (.jar), graphic interchange format (.gif), cascading style sheet (.css), audio files (.wav) and extensible markup language (.xml) files.

1.11 "Sample Application Code" means Software provided for the purposes of demonstrating functionality of an Avaya product through the Avaya Software Development Kit.

1.12 "Software" means data or information constituting one or more computer or apparatus programs, including Source Code or in machine-readable, compiled object code form.

2.0 LICENSE GRANT.

2.1 SDK License.

A. Provided Licensee pays to Avaya the applicable license fee (if any), Avaya hereby grants Licensee a limited, non-exclusive, non-transferable license (without the right to sublicense, except as set forth in 2.1B(iii)) under the Intellectual Property of Avaya and, if applicable, its licensors and suppliers to (i) use the SDK solely for the purpose of Licensee's internal development efforts to develop applications, interfaces, value-added services and/or solutions, workflows or processes to work in conjunction with Avaya products; (ii) to package Client Libraries for redistribution with Licensee's complementary applications that have been developed using this SDK, subject to the terms and conditions set forth herein; (iii) use Specification Documents solely to enable Licensee's products, services and application solutions to exchange messages and signals with Avaya products, systems and solutions to which the Specification Document(s) apply; (iv) modify and create Derivative Works of the Sample Application Code, Specification Documents and Documentation solely for internal development of applications, interfaces, workflows or processes for use with Avaya products, integration of such applications, interfaces, workflows and processes with Avaya products and interoperability testing of the foregoing with Avaya products; and (v) compile or otherwise prepare for distribution the Sample Application Code with Permitted Modifications, into an object code or other machine-readable program format for distribution and distribute the same subject to the conditions set forth in Section 2.1B.

B. The foregoing license to use Sample Application Code is contingent upon the following: (i) Licensee must ensure that the modifications made to the Sample Application Code as permitted in clause (iv) of Section 2.1A are compatible and/or interoperable with Avaya products and/or integrated therewith, (ii) Licensee may distribute Licensee's application that has been created using this SDK, provided that such distribution is subject to an end user pursuant to Licensee's current end user license agreement ("Licensee EULA") that is consistent with the terms of this Agreement and, if applicable, any other agreement with Avaya (e.g., the Avaya DevConnect Program Agreement), and is equally as protective as Licensee's standard software license terms, but in no event shall the standard of care be less than a reasonable degree of care, and (iii) Licensee ensures that each end user who receives Client Libraries or Sample Application Code with Permitted Modifications has all necessary licenses for all underlying Avaya products associated with such Client Libraries or Sample Application Code.

Your Licensee EULA must include terms concerning restrictions on use, protection of proprietary rights, disclaimer of warranties, and limitations of liability. You must ensure that Your End Users using applications, interfaces, value-added services and/or solutions, workflows or processes that incorporate the API, Client Libraries, Sample Code or Permitted Modifications adhere to these terms, and You agree to notify Avaya promptly if You become aware of any breach of the terms of Licensee EULA that may impact Avaya. You will take all reasonable precautions to prevent unauthorized access to or use of the SDK and notify Avaya promptly of any such unauthorized access or use.

C. Licensee acknowledges and agrees that it is licensed to use the SDK only in connection with Avaya products (and if applicable, in connection with services provided by or on behalf of Avaya).

D. With respect to Software that contains elements provided by third party suppliers, Licensee may install and use the Software in accordance with the terms and conditions of the applicable license agreements, such as "shrinkwrap" or "click-through" licenses, accompanying or applicable to the Software.

2.2 No Standalone Product. Nothing in this Agreement authorizes or grants Licensee any rights to distribute or otherwise make available to a third party the SDK, in whole or in part, or any Derivative Work in source or object code format on a standalone basis other than the modifications permitted in Section 2.1B of this Agreement.

2.3 Proprietary Notices. Licensee shall not remove any copyright, trade mark or other proprietary notices incorporated in the copies of the SDK, Sample Application Code and redistributable files in Licensee's possession or control or any modifications thereto. Redistributions in binary form or other suitable program format for distribution, to the extent expressly permitted, must also reproduce Avaya's copyright, trademarks or other proprietary notices as incorporated in the SDK in any associated Documentation or "splash screens" that display Licensee copyright notices.

2.4 Third-Party Components. You acknowledge certain software programs or portions thereof included in the SDK may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the SDK ("Third Party Terms"). Information identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the attached Schedule 1 (if any), SDK, Documentation, or on Avaya's web site at: <http://support.avaya.com/Copyright> (or such successor site as designated by Avaya). The open source software license terms provided as Third Party Terms are consistent with the license rights granted in this Agreement, and may contain additional rights benefiting You, such as modification and distribution of the open source software. The Third Party Terms shall take precedence over this Agreement, solely with respect to the applicable Third Party Components, to the extent that this Agreement imposes greater restrictions on You than the applicable Third Party Terms. Licensee is solely responsible for procuring any necessary licenses for Third Party Components, including payment of licensing royalties or other amounts to third parties, for the use thereof.

2.5 Copies of SDK. Licensee may copy the SDK only as necessary to exercise its rights hereunder.

2.6a No Reverse Engineering. Licensee shall have no rights to any Source Code for any of the software in the SDK, except for the explicit rights to use the Source Code as provided to Licensee hereunder. Licensee agrees that it shall not cause or permit the disassembly, decompilation or reverse engineering of the Software. Notwithstanding the foregoing, if the SDK is rightfully located in a member state of the European Union and Licensee needs information about the Software in the SDK in order to achieve interoperability of an independently created software program with the Software in the SDK, Licensee will first request such information from Avaya. Avaya may charge Licensee a reasonable fee for the provision of such information. If Avaya refuses to make such information available, then Licensee may take steps, such as reverse assembly or reverse compilation, to the extent necessary solely in order to achieve interoperability of the Software in the SDK with an independently created software program. To the extent that the Licensee is expressly permitted by applicable mandatory law to undertake any of the activities listed in this section, Licensee will not exercise those rights until Licensee has given Avaya twenty (20) days written notice of its intent to exercise any such rights.

2.6.b License Restrictions. To the extent permissible under applicable law, Licensee agrees not to: (i) publish, sell, sublicense, lease, rent, loan, assign, convey or otherwise transfer the SDK; (ii) distribute, disclose or allow use the SDK, in any format, through any timesharing service, service bureau, network or by any other means; (iii) distribute or otherwise use the Software in the SDK in any manner that causes any portion of the Software that is not already subject to an OSS License to become subject to the terms of any OSS License; (iv) link the Source Code for any of the software in the SDK with any software licensed under the Affero General Public License (Affero GPL) v.3 or similar licenses; (v) access information that is solely available to root administrators of the Avaya products, systems, and solutions; (vi) develop applications, interfaces, value-added services and/or solutions, workflows or processes that causes adverse effects to Avaya and third-party products, services, solutions, such as, but not limited to, poor performance, software crashes and cessation of their proper functions; and (vii) develop applications, interfaces, value-added services and/or solutions, workflows or processes that blocks or delays emergency calls; (viii) emulate an Avaya SIP endpoint by form or user interface design confusingly similar as an Avaya product; (ix) reverse engineer Avaya SIP protocol messages; or (x) permit or encourage any third party to do any of (i) through (x), inclusive, above.

2.7 Responsibility for Development Tools. Licensee acknowledges that effective utilization of the SDK may require the use of a development tool, compiler and other software and technology of third parties, which may be incorporated in the SDK pursuant to Section 2.4. Licensee is solely responsible for procuring such third party software and technology and the necessary licenses, including payment of licensing royalties or other amounts to third parties, for the use thereof.

2.8 U.S. Government End Users. The SDK shall be classified as "commercial computer software" and the Documentation is classified as "commercial computer software documentation" or "commercial items," pursuant to FAR 12.212 or DFAR 227.7202, as applicable. Any use, modification, reproduction, release, performance, display or disclosure of the SDK or Documentation by the Government of the United States shall be governed solely by the terms of the Agreement and shall be prohibited except to the extent expressly permitted by the terms of the Agreement.

2.9 Limitation of Rights. No right is granted to Licensee to sublicense its rights hereunder. All rights not expressly granted are reserved by Avaya or its licensors or suppliers and, except as expressly set forth herein, no license is granted by Avaya or its licensors or suppliers under this Agreement directly, by implication, estoppel or otherwise, under any Intellectual Property right of Avaya or its licensors or suppliers. Nothing herein shall be deemed to authorize Licensee to use Avaya's trademarks or trade names in Licensee's advertising, marketing, promotional, sales or related materials.

2.10 Independent Development.

2.10.1 Licensee understands and agrees that Avaya, Affiliates, or Avaya's licensees or suppliers may acquire, license, develop for itself or have others develop for it, and market and/or distribute applications, interfaces, value-added services and/or solutions, workflows or processes similar to that which Licensee may develop. Nothing in this Agreement shall restrict or limit the rights of Avaya, Affiliates, or Avaya's licensees or suppliers to commence or continue with the development or distribution of such applications, interfaces, value-added services and/or solutions, workflows or processes.

2.10.2 Nonassertion by Licensee. Licensee agrees not to assert any Intellectual Property related to the SDK or applications, interfaces, value-added services and/or solutions, workflows or processes developed using the SDK against Avaya, Affiliates, Avaya's licensors or suppliers, distributors, customers, or other licensees of the SDK.

2.11 Feedback and Support. Licensee agrees to provide any information, comments, problem reports, enhancement requests and suggestions regarding the performance of the SDK (collectively, "Feedback") via any public or private support mechanism, forum or process otherwise indicated by Avaya. Avaya monitors applicable mechanisms, forums, or processes but is under no obligation to implement any of Feedback, or be required to respond to any questions asked via the applicable mechanism, forum, or process. Licensee hereby assigns to Avaya all right, title, and interest in and to Feedback provided to Avaya.

2.12(a) Fees and Taxes. To the extent that fees are associated with the license of the SDK, Licensee agrees to pay to Avaya or pay directly to the applicable government or taxing authority, if requested by Avaya, all taxes and charges, including without limitation, penalties and interest, which may be imposed by any federal, state or local governmental or taxing authority arising hereunder excluding, however, all taxes computed upon Avaya's net income. If You move any Software, including the SDK, and as a result of such move, a jurisdiction imposes a duty, tax, levy or fee (including withholding taxes, fees, customs or other duties for the import and export of any such Software), then You are solely liable for, and agree to pay, any such duty, taxes, levy or other fees.

2.12(b) Audit. Avaya shall have the right, at its cost and expense, to inspect and/or audit (i) by remote polling or other reasonable electronic means at any time and (ii) in person during normal business hours and with reasonable notice Licensee's books, records, and accounts, to determine Licensee's compliance with this Agreement. In the event such inspection or audit uncovers non-compliance with this Agreement, then without prejudice to Avaya's termination rights hereunder, Licensee shall promptly pay Avaya any applicable license fees. Licensee agrees to keep a current record of the location of the SDK.

2.13 No Endorsement. Neither the name Avaya, Affiliates nor the names of contributors may be used to endorse or promote products derived from the Avaya SDK without specific prior written permission from Avaya.

2.14 High Risk Activities. The Avaya SDK is not fault-tolerant, and is not designed, manufactured or intended for use or resale as on-line control equipment or in hazardous environments requiring failsafe performance, such as in the operation of nuclear facilities, aircraft navigation or aircraft communications systems, mass transit, air traffic control, medical or direct life support machines, dedicated emergency call handling systems or weapons systems, in which the failure of the Avaya SDK could lead directly to death, personal injury, or severe physical or environmental damage ("high risk activities"). If Licensee uses the Avaya SDK for high risk activities, Licensee does so at Licensee's own risk and Licensee assumes all responsibility and liability for such use to the maximum extent such limitation or exclusion is permitted by applicable law. Licensee agrees that Avaya and its suppliers will not be liable for any claims or damages arising from or related to use of the Avaya SDK for high risk activities to the maximum extent such limitation or exclusion is permitted by law.

2.15 No Virus. Licensee warrants that (i) the applications, interfaces, value-added services and/or solutions, workflows or processes Licensee develops using this SDK will not contain any computer program file that includes time code limitations, disabling devices, or any other mechanism which will prevent the Avaya product (including other software, firmware, hardware), services and networks from being functional at all times (collectively "Time Bombs"); and (ii) the applications, interfaces, value-added services and/or solutions, workflows or processes Licensee develops using this SDK will be free of computer viruses, malicious or other harmful code, black boxes, malware, trapdoors, and other mechanisms which could: a) damage, destroy or adversely affect Avaya product, or services and/or end users; b) allow remote/hidden attacks or access through unauthorized computerized command and control; c) spy (network sniffers, keyloggers), and d) damage or erase such applications, interfaces, value-added services and/or solutions, workflows or processes developed using this SDK or data, or any computer files or systems of Avaya, Affiliates, and/or end users (collectively "Virus"). In addition to any other remedies permitted in the Agreement, if Licensee breaches its warranties under this Section, Licensee will, at its expense, take remedial action to eliminate any Time Bombs and/or Viruses and prevent re-occurrence (including implementing appropriate processes to prevent further occurrences) as well as provide prompt, reasonable assistance to Avaya to materially reduce the effects of the Time Bomb and/or Virus.

2.16 Disclaimer. Any software security feature is not a guaranty against malicious code, deleterious routines, and other techniques and tools employed by computer "hackers" and other third parties to create security exposures. Compromised passwords represent a major security risk. Avaya encourages You to create strong passwords using three different character types, change Your password regularly and refrain from using the same password regularly. You must treat such information as confidential. You agree to notify Avaya immediately upon becoming aware of any unauthorized use or breach of Your user name, password, account, API Key, or other credentials as provided by Avaya for use of the SDK, or subscription. You are responsible for ensuring that Your networks and systems are adequately secured against unauthorized intrusion or attack and regularly back up of Your data and files in accordance with good computing practices.

2.17 Third Party Licensed Software

A. "Commercial Third Party Licensed Software" is software developed by a business with the purpose of making money from the use of that licensed software. "Freeware Licensed Software" is software which is made available for use, free of charge and for an unlimited time, but is not Open Source Licensed Software. "Open Source Software" or "OSS" is as defined by the Open Source Initiative ("OSI") <https://opensource.org/osd> and is software licensed under an OSI approved license as set forth at <https://opensource.org/licenses/alphabetical> (or such successor site as designated by OSI). These are collectively referred to herein as "Third Party Licensed Software".

B. Licensee represents and warrants that Licensee, including any employee, contractor, subcontractor, or consultant engaged by Licensee, is to the Licensee's knowledge, in compliance and will continue to comply with all license obligations for Third Party Licensed Software used in the Licensee application created using the SDK including providing to end users all information required by such licenses as may be necessary. LICENSEE REPRESENTS AND WARRANTS THAT, TO THE LICENSEE'S KNOWLEDGE, THE OPEN SOURCE LICENSED SOFTWARE EMBEDDED IN OR PROVIDED WITH LICENSEE APPLICATION OR SERVICES DOES NOT INCLUDE ANY OPEN SOURCE LICENSED SOFTWARE CONTAINING TERMS REQUIRING ANY INTELLECTUAL PROPERTY OWNED OR LICENSED BY AVAYA OR END USERS TO BE (A) DISCLOSED OR DISTRIBUTED IN SOURCE CODE OR OBJECT CODE FORM; (B) LICENSED FOR THE PURPOSE OF MAKING DERIVATIVE WORKS; OR (C) REDISTRIBUTABLE ON TERMS AND CONDITION NOT AGREED UPON BY AVAYA OR END USERS.

1. Subject to any confidentiality obligations, trade secret or other rights or claims of Licensee suppliers, Licensee will respond to requests from Avaya or end users relating to Third Party Licensed Software associated with Licensee's use of Third Party Licensed Software. Licensee will cooperate in good faith by furnishing the relevant information to Avaya or end users and the requester within two (2) weeks from the time Avaya or end user provided the request to Licensee.

3. OWNERSHIP.

3.1 As between Avaya and Licensee, Avaya or its licensors or suppliers shall own and retain all Intellectual Property rights, in and to the SDK and any corrections, bug fixes, enhancements, updates, improvements, or modifications thereto and Licensee hereby irrevocably transfers, conveys and assigns to Avaya, its licensors and its suppliers all of its right, title, and interest therein. Avaya or its licensors or suppliers shall have the exclusive right to apply for or register any patents, mask work rights, copyrights, and such other proprietary protections with respect thereto. Licensee acknowledges that the license granted under this Agreement does not provide Licensee with title or ownership to the SDK, but only a right of limited use under the terms and conditions of this Agreement.

3.2 Grant Back License to Avaya. Licensee hereby grants to Avaya an irrevocable, perpetual, non-exclusive, sublicensable, royalty-free, fully paid up, worldwide license under any and all of Licensee's Intellectual Property rights related to any Permitted Modifications, to (i) use, make, sell, execute, adapt, translate, reproduce, display, perform, prepare derivative works based upon, distribute (internally and externally) and sublicense the Permitted Modifications and their derivative works, and (ii) sublicense others to do any, some, or all of the foregoing.

4.0 SUPPORT.

4.1 No Avaya Support. Avaya will not provide any support for the SDK provided under this Agreement or for any Derivative Works, including, without limitation, modifications to the Source Code or applications built by Licensee using the SDK. Avaya shall have no obligation to provide support for the use of the SDK, or Licensee's application, services or solutions which may or may not include redistributable Client Libraries or Sample Application Code, to any third party to whom Licensee delivers such applications, services or solutions. Avaya further will not provide fixes, patches or repairs for any defects that might exist in the SDK or the Sample Application Code provided under this Agreement. In the event that Licensee desires support services for the SDK, and, provided that Avaya offers such support services (in its sole discretion), Licensee will be required to enter into an Avaya DevConnect Program Agreement or other support agreement with Avaya.

4.2 Licensee Obligations. Licensee acknowledges and agrees that it is solely responsible for developing and supporting any applications, interfaces, value-added services and/or solutions, workflows or processes developed under this Agreement, including but not limited to (i) developing, testing and deploying such applications, interfaces, value-added services and/or solutions, workflows or processes; (ii) configuring such applications, interfaces, value-added services and/or solutions, workflows or processes to interface and communicate properly with Avaya products; and (iii) updating and maintaining such applications, interfaces, value-added services and/or solutions, workflows or processes as necessary for continued use with the same or different versions of end user and/or third party licensor products, and Avaya products.

5.0 CONFIDENTIALITY.

5.1 Protection of Confidential Information. Licensee acknowledges and agrees that the SDK and any other Avaya technical information obtained by it under this Agreement (collectively, "Confidential Information") is confidential information of Avaya. Licensee shall take all reasonable measures to maintain the confidentiality of the Confidential Information. Licensee further agrees at all times to protect and preserve the SDK in strict confidence in perpetuity, and shall not use such Confidential Information other than as expressly authorized by Avaya under this Agreement, nor shall Licensee disclose any Confidential Information to third parties without Avaya's written consent. Licensee further agrees to immediately 1) cease all use of all Confidential Information (including copies thereof) in Licensee's possession, custody, or control; 2) stop reproducing or distributing the Confidential Information; and 3) destroy the Confidential Information in Licensee's possession or under its control, including Confidential Information on its computers, disks, and other digital storage devices upon termination of this Agreement at any time and for any reason. Upon request, Licensee will certify in writing its compliance with this Section. The obligations of confidentiality shall not apply to information which (a) has entered the public domain except where such entry is the result of Licensee's breach of this Agreement; (b) prior to disclosure hereunder was already rightfully in Licensee's possession; (c) subsequent to disclosure hereunder is obtained by Licensee on a non-confidential basis from a third party who has the right to disclose such information to the Licensee; (d) is required to be disclosed pursuant to a court order, so long as Avaya is given adequate notice and the ability to challenge such required disclosure.

5.2 Press Releases. Any press release or publication regarding this Agreement is subject to prior written approval of Avaya.

6.0 NO WARRANTY.

The SDK and Documentation are provided "AS-IS" without any warranty whatsoever. AVAYA SPECIFICALLY AND EXPRESSLY DISCLAIMS ANY WARRANTIES OR CONDITIONS, STATUTORY OR OTHERWISE, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT AND SATISFACTORY QUALITY. AVAYA DOES NOT WARRANT THAT THE SDK AND DOCUMENTATION ARE SUITABLE FOR LICENSEE'S USE, THAT THE SDK OR DOCUMENTATION ARE WITHOUT DEFECT OR ERROR, THAT OPERATION WILL BE UNINTERRUPTED, OR THAT DEFECTS WILL BE CORRECTED. FURTHER, AVAYA MAKES NO WARRANTY REGARDING THE RESULTS OF THE USE OF THE SDK AND DOCUMENTATION. NEITHER AVAYA NOR ITS SUPPLIERS MAKE ANY WARRANTY, EXPRESS OR IMPLIED, THAT THE SDK OR DOCUMENTATION IS SECURE, SECURITY THREATS AND VULNERABILITIES WILL BE DETECTED OR SOFTWARE WILL RENDER AN END USER'S OR LICENSEE'S NETWORK OR PARTICULAR NETWORK ELEMENTS SAFE FROM INTRUSIONS AND OTHER SECURITY BREACHES.

7.0 CONSEQUENTIAL DAMAGES WAIVER.

EXCEPT FOR PERSONAL INJURY CLAIMS, AVAYA SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH, ARISING OUT OF OR RELATING TO THIS AGREEMENT OR USE OF THE SDK, OR FOR THE LOSS OR CORRUPTION OF DATA, INFORMATION OF ANY KIND, BUSINESS, PROFITS, OR OTHER COMMERCIAL LOSS, HOWEVER CAUSED, AND WHETHER OR NOT AVAYA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8.0 LIMITATION OF LIABILITY.

EXCEPT FOR PERSONAL INJURY CLAIMS, IN NO EVENT SHALL AVAYA'S TOTAL LIABILITY TO LICENSEE IN CONNECTION WITH, ARISING OUT OF OR RELATING TO THIS AGREEMENT EXCEED FIVE HUNDRED DOLLARS (\$500). THE PARTIES AGREE THAT THE LIMITATIONS SPECIFIED IN THIS SECTION WILL APPLY EVEN IF ANY LIMITED REMEDY PROVIDED IN THIS AGREEMENT IS FOUND TO HAVE FAILED OF ITS ESSENTIAL PURPOSE.

9.0 INDEMNIFICATION.

Licensee shall indemnify and hold harmless Avaya, Affiliates and their respective officers, directors, agents, suppliers, customers and employees ("Indemnified Parties") from and against all claims, demand, suit, actions or proceedings ("Claims") and damages, losses, liabilities, costs, expenses, and fees (including fees of attorneys and other professionals) ("Damages") based upon an allegation pertaining to wrongful use, misappropriation, or infringement of a third party's Intellectual Property right arising from or relating to Licensee's use of the SDK, alone or in combination with other software, such as operating systems and codecs, and the, direct or indirect, use, distribution or sale of any software, Derivative Works or other products (including but not limited to applications, interfaces, and application programming interfaces) developed utilizing the SDK.

Licensee shall defend, indemnify and hold harmless the Indemnified Parties from and against all Claims and Damages arising out of or related to: (i) personal injury (including death); (ii) damage to any person or tangible property caused, or alleged to be caused by Licensee or Licensee's application created by using the SDK; (iii) the failure by Licensee or Licensee's application created by using the SDK to comply with the terms of this Agreement or any applicable laws; (iv) the breach of any representation, or warranty made by Licensee herein; or (v) Licensee's breach of any obligation under the Licensee EULA.

10.0 TERM AND TERMINATION.

10.1 This Agreement will continue through December 31st of the current calendar year. The Agreement will automatically renew for one (1) year terms, unless terminated as specified in Section 10.2 or 10.3 below.

10.2 Either party shall have the right to terminate the Agreement, upon thirty (30) days written notice to the other party.

10.3 Notwithstanding language to the contrary, Avaya may terminate this Agreement immediately, upon written notice to Licensee for breach of Section 2 (License Grant), Section 5 (Confidentiality) or Section 12 (Compliance with Laws). Avaya may also terminate this Agreement immediately by giving written notice if a Change In Control should occur or if Licensee becomes insolvent, or voluntary or involuntary proceedings by or against Licensee are instituted in bankruptcy or under any insolvency law, or a receiver or custodian is appointed for Licensee, or proceedings are instituted by or against Licensee for corporate reorganization or the dissolution of Licensee, which proceedings, if involuntary, have not been dismissed within thirty (30) days after the date of filing, or Licensee makes an assignment for the benefit of its creditors, or substantially all of the assets of Licensee are seized or attached and not released within sixty (60) days thereafter, or if Licensee has ceased or threatened to cease to do business in the regular course.

10.4 Upon termination or earlier termination of this Agreement, Licensee will immediately cease a) all uses of the Confidential Information; b) Licensee agrees to destroy all adaptations or copies of the Confidential Information stored in any tangible medium including any document or work containing or derived (in whole or in part) from the Confidential Information, and certify its destruction to Avaya upon termination of this License. Licensee will promptly cease use of, distribution and sales of Licensee products that embody any such Confidential Information, and destroy all Confidential Information belonging to Avaya as well as any materials that embody any such Confidential Information. All licenses granted will terminate.

10.5 The rights and obligations of the parties contained in Sections 2.3, 2.6, 2.7, 2.10, 2.11, 2.12, 3, and 5 through 18 shall survive any expiration or termination of this Agreement.

11.0 ASSIGNMENT.

Avaya may assign all or any part of its rights and obligations hereunder. Licensee may not assign this Agreement or any interest or rights granted hereunder to any third party without the prior written consent of Avaya. The term "assign" includes, but is not limited to, any transaction in which there is a Change In Control or reorganization of Licensee pursuant to a merger, sale of assets or stock. This Agreement shall terminate immediately upon occurrence of any prohibited assignment.

12.0 COMPLIANCE WITH LAWS.

Licensee shall comply with all applicable laws and regulations, including without limitation those applicable to data privacy, intellectual property, trade secret, fraud, music performance rights and the export or re-export of technology and will not export or re-export the SDK or any other technical information provided under this Agreement in any form in violation of the export control laws of the United States of America and of any other applicable country. For more information on such export laws and regulations, Licensee may refer to the resources provided in the websites maintained by the U.S. Commerce Department, the U.S. State Department and the U.S. Office of Foreign Assets Control.

13.0 WAIVER.

The failure to assert any rights under this Agreement, including, but not limited to, the right to terminate in the event of breach or default, will not be deemed to constitute a waiver of the right to enforce each and every provision of this Agreement in accordance with their terms.

14.0 SEVERABILITY.

If any provision of this Agreement is determined to be unenforceable or invalid, this Agreement will not be rendered unenforceable or invalid as a whole, and the provision will be changed and interpreted so as to best accomplish the objectives of the original provision within the limits of applicable law.

15.0 GOVERNING LAW AND DISPUTE RESOLUTION.

15.1 Governing Law. This Agreement and any dispute, claim or controversy arising out of or relating to this Agreement ("Dispute"), including without limitation the formation, interpretation, breach or termination of this Agreement, or any issue regarding whether a Dispute is subject to arbitration under this Agreement, will be governed by New York State laws, excluding conflict of law principles, and the United Nations Convention on Contracts for the International Sale of Goods.

15.2 Dispute Resolution. Any Dispute will be resolved in accordance with the provisions of this Section 15. The disputing party shall give the other party written notice of the Dispute in accordance with the notice provision of this Agreement. The parties will attempt in good faith to resolve each controversy or claim within 30 days, or such other longer period as the parties may mutually agree, following the delivery of such notice, by negotiations between designated representatives of the parties who have dispute resolution authority.

15.3 Arbitration of Non-US Disputes. If a Dispute that arose anywhere other than in the United States or is based upon an alleged breach committed anywhere other than in the United States cannot be settled under the procedures and within the timeframe set forth in Section 15.2, it will be conclusively determined upon request of either party by a final and binding arbitration proceeding to be held in accordance with the Rules of Arbitration of the International Chamber of Commerce by a single arbitrator appointed by the parties or (failing agreement) by an arbitrator appointed by the President of the International Chamber of Commerce (from time to time), except that if the aggregate claims, cross claims and counterclaims by any one party against the other party exceed One Million US Dollars at the time all claims, including cross claims and counterclaims are filed, the proceeding will be held in accordance with the Rules of Arbitration of the International Chamber of Commerce by a panel of three arbitrator(s) appointed in accordance with the Rules of Arbitration of the International Chamber of Commerce. The arbitration will be conducted in the English language, at a location agreed by the parties or (failing agreement) ordered by the arbitrator(s). The arbitrator(s) will have authority only to award compensatory damages within the scope of the limitations of Section 8 and will not award punitive or exemplary damages. The arbitrator(s) will not have the authority to limit, expand or otherwise modify the terms of this Agreement. The ruling by the arbitrator(s) will be final and binding on the parties and may be entered in any court having jurisdiction over the parties or any of their assets. The parties will evenly split the cost of the arbitrator(s)' fees, but Avaya and Customer will each bear its own attorneys' fees and other costs associated with the arbitration. The parties, their representatives, other participants and the arbitrator(s) will hold the existence, content and results of the arbitration in strict confidence to the fullest extent permitted by law. Any disclosure of the existence, content and results of the arbitration will be as limited and narrowed as required to comply with the applicable law. By way of illustration, if the applicable law mandates the disclosure of the monetary amount of an arbitration award only, the underlying opinion or rationale for that award may not be disclosed.

15.4 Choice of Forum for US Disputes. If a Dispute by one party against the other that arose in the United States or is based upon an alleged breach committed in the United States cannot be settled under the procedures and within the timeframe set forth in Section 15.2, then either party may bring an action or proceeding solely in either the Supreme Court of the State of New York, New York County, or the United States District Court for the Southern District of New York. Except as otherwise stated in Section 15.3 each party consents to the exclusive jurisdiction of those courts, including their appellate courts, for the purpose of all actions and proceedings arising out of or relating to this Agreement.

15.5 Injunctive Relief. Nothing in this Agreement will be construed to preclude either party from seeking provisional remedies, including, but not limited to, temporary restraining orders and preliminary injunctions from any court of competent jurisdiction in order to protect its rights, including its rights pending arbitration, at any time. The parties agree that the arbitration provision in Section 15.3 may be enforced by injunction or other equitable order, and no bond or security of any kind will be required with respect to any such injunction or order.

15.6 Time Limit. Actions on Disputes between the parties must be brought in accordance with this Section within 2 years after the cause of action arises.

16.0 IMPORT/EXPORT CONTROL.

Licensee is advised that the SDK is of U.S. origin and subject to the U.S. Export Administration Regulations ("EAR"). The SDK also may be subject to applicable local country import/export laws and regulations. Diversion contrary to U.S. and/or applicable local country law and/or regulation is prohibited. Licensee agrees not to directly or indirectly export, re-export, import, download, or transmit the SDK to any country, end user or for any use that is contrary to applicable U.S. and/or local country regulation or statute (including but not limited to those countries embargoed by the U.S. government). Licensee represents that any governmental agency has not issued sanctions against Licensee or otherwise suspended, revoked or denied Licensee's import/export privileges. Licensee agrees not to use or transfer the SDK for any use relating to nuclear, chemical or biological weapons, or missile technology, unless authorized by the U.S. and/or any applicable local government by regulation or specific written license. Additionally, Licensee is advised that the SDK may contain encryption algorithm or source code that may not be exported to government or military end users without a license issued by the U.S. Bureau of Industry and Security and any other country's governmental agencies, where applicable.

17.0 AGREEMENT IN ENGLISH.

The parties confirm that it is their wish that the Agreement, as well as all other documents relating hereto, including all notices, have been and shall be drawn up in the English language only. Les parties aux présentes confirment leur volonté que cette convention, de même que tous les documents, y compris tout avis, qui s'y rattachent, soient rédigés en langue anglaise.

18.0 ENTIRE AGREEMENT.

This Agreement, its exhibits, schedules and other agreements or documents referenced herein, constitute the full and complete understanding and agreement between the parties and supersede all contemporaneous and prior understandings, agreements and representations relating to the subject matter hereof. No modifications, alterations or amendments shall be effective unless in writing signed by both parties to this Agreement.

19. Redistributable Client Files.

The list of SDK client files that can be redistributed, if any, are in the SDK in a file called Redistributable.txt.

Schedule 1 to Avaya SDK License Agreement Third Party Notices

- 1. CODECS:** WITH RESPECT TO ANY CODECS IN THE SDK, YOU ACKNOWLEDGE AND AGREE YOU ARE RESPONSIBLE FOR ANY AND ALL RELATED FEES AND/OR ROYALTIES, IF ANY. IT IS YOUR RESPONSIBILITY TO CHECK.

THE H.264 (AVC) CODEC IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO: (I) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD ("AVC VIDEO") AND/OR (II) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION FOR THE H.264 (AVC) CODEC MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE [HTTP://WWW.MPEGLA.COM](http://www.mpegla.com).

Purpose and Intended Audience

This guide is aimed at developers who are creating a customer-facing website for an Avaya Oceana chat solution. Its purpose is to aid deploying, configuring and troubleshooting the Web UI and the sample native Android app.

Disclaimer

The chat and Android clients are samples that are provided as a reference implementation. Avaya is not responsible for customising this to match your organisational needs.

The Web UI

The Web UI is a sample reference website built around a JavaScript library that opens a chat to the Customer Controller component in Oceana. The library itself consumes jQuery 3.3.1 and jQuery UI 1.12.4, the AvayaClientServices library, and Bootstrap 4.0.0.

1. Supported Browsers

The Web UI has been tested on the following browsers:

- Firefox (any version released in 2017 or later)
- Chrome (any version released in 2017 or later)
- Internet Explorer 11
- Microsoft Edge 38 and later.

Other browsers may work, but have not been tested. The following browser features are **required**:

- WebSockets
- XMLHttpRequest
- JavaScript strict mode

2. Deploying the Web UI

The Web UI should be deployed to a server outside the Breeze network. This has been tested on Apache HTTP 2.4.25, and this step-by-step guide assumes you have an Apache server installed. Installing Apache is beyond the scope of this guide. Consult the [Apache documentation](#) for further information.

1. Extract the WAR or zip file to a folder on e.g. the desktop. This can be done using WinZip, 7-Zip, tar or similar file compression programmes.
2. Under the Apache root directory (henceforth referred to as **\${SRVROOT}**), find the htdocs folder.
3. Create a new folder under this for the Web UI. The path should be similar to **\${SRVROOT}/htdocs/webui**.
4. Copy the contents of the **src/main/webapp** folder to this folder.
5. Open the Apache configuration file (**\${SRVROOT}/conf/httpd.conf**), and define a virtual host as in the following example, if there is not already one that listens on port 80. Restart Apache afterwards to make sure it has been configured properly.

```
# Instruct Apache to list on port 80.
Listen 80

# Create a VirtualHost on port 80.
# This one applies to all sites hosted on this port.
<VirtualHost *:80>
    ServerName apacheServer
    DirectoryIndex home.html
    DocumentRoot "${SRVROOT}/htdocs/webui"
</VirtualHost>
```

6. Open the **src/main/webapp/js/links.js** file. Change the URLs as follows:

- Change the IP address or hostname in **webChatHost** to point at the Omni-Channel cluster.
- Change the IP address or hostname in **contextStoreHost** to point at the cluster that hosts Avaya Oceana Core Data Service.
- Change the IP address or hostname in **coBrowseHost** to point at the Co-Browsing server.
- Change the **secureAllConnections** variable to true to use a secure connection. If the page is served over HTTPS, this will be automatically set, as modern browsers forbid HTTP connections from a page that is served over HTTPS.
- If using a reverse proxy to hide Oceana from the wider Internet, configure each URL to point at the corresponding URL on the proxy.



Changes to the Web UI do not require Apache to be restarted.

3. Configuring the Web UI

3.1. Entry points

The following table indicates where to start looking.

Method	File	Purpose	When is it called (by default)	How is it called
setupWebChat	webChat.js	Set up required methods etc for web chat	On page load	Inline script inside home.html
setup	webChatUI.js	Sets up UI-specific code (panels, widgets, etc)	Called when <i>setupWebChat</i> method is called	Direct function call
gatherDetails	webChatLogon.js	Gathers customer details and kicks off a chat	Clicking the "Chat Now" button	Inline function call
setup	customerJourneyUI.js	Set up the UI elements for the customer journey	On page load	Inline function call from page
setup	customerJourney.js	Set up the customer journey code	From customerJourneyUI <i>setup</i> function, if customer already subscribed	Direct function call
requestEwt	estimatedWaitTime.js	Request estimated wait time	Called when <i>setupWebChat</i> method is called	Direct function call
handleMessage	webChatSocket.js	Implements the WebSocket onmessage handler. Acts as the entry point for receiving chat messages	Assigned from the <i>openSocket</i> method inside webChatSocket.js. Invoked in background by browser	Background method by browser)

3.2. Third-Party libraries

The out-of-the-box UI uses the following libraries:

Library name	Version	Where it's used	Purpose
jQuery	3.3.1	webChatUI.js, coBrowseUI.js, contextStoreUI.js	Shorthand for DOM interactions and initialisation
jQuery UI	1.12.1	webChatUI.js, coBrowseUI.js, contextStoreUI.js	Creating draggable widgets/panels
Bootstrap	4.0.0	HTML pages	Page layout
AvayaCoBrowseClientServices	Aligned with Oceana	coBrowse.js	Cobrowsing interactions. This is developed as a separate internal library

3.3. Using the configuration panel

A configuration panel has been added to the home.html page to allow you to configure the UI for a lab demo. Everything here is saved to localStorage, allowing it to persist over page refreshes. This section details how to use it:

1. Open the home.html page in a browser.
2. Click the gear icon in the top-right corner. The panel will appear.
3. Click on the first line ("Click here to configure URLs") to open/close the URL configuration section. Change the hostnames/IP addresses for the chat/estimated wait time cluster, the Customer Journey cluster, and the Co-Browsing cluster. This does not save automatically.
4. Click on the second line ("Click here to configure attributes") to open/close the chat attribute configuration section.
5. To add a new attribute, enter its value into the textbox at the top of the section, and click "Add". This will add the value to the chat attributes.
6. To remove an attribute, click the "Remove" button next to its entry. This will automatically remove the attribute from the chat attributes.
7. Click on the third line ("Click here to configure workflow and routepoints") to open the workflow and routepoint configuration section.
8. Change the workflow type and routepoint to match your settings.
9. Click the "Save" button to save the configuration.
10. To reset the configuration to the default, click the "Reset" button and reload the page.
11. To request a new estimated wait time for the chat attributes, click the "Request Estimated Wait Time" button.
12. If you need to enter a chat regardless of the estimated wait time, click the "Show Chat Panel Anyway" button.
13. To initialise the Co-Browsing library, click the "Initialise Co-Browse Library" button.



The configuration panel is **not** a deployment mechanism. Any changes made using this will **not** apply to other users, or to another browser on the same machine.

3.4. Secure Connections

Secure connections from the Web UI require generating certificates for the Omni-Channel and Co-Browsing clusters. This section details how to do this using the default certificates from System Manager. If you are using your own certificates from a third-party certification authority, refer to the Avaya Oceana documentation. Step 1 may be skipped in this case.

1. Open System Manager.
2. Click "Security" in the right-hand menu.
3. Click "Certificates", then click "Authority".
4. Click on "Public Web". This will open a public page to retrieve certificates.
5. Click "Fetch CA Certificates" in the menu along the left-hand-side.
6. Click "Download to Internet Explorer" to install for Internet Explorer, Edge and Chrome. Click "Download to Firefox" to install specifically to Firefox.
 - a. There is also a configuration flag in Firefox to force it to use the operating system's certificate manager. To toggle this, visit the [about:config](#) page in Firefox, and set the value of "security.enterprise_roots" to **true** (the default value is **false**).
7. Change the "useSecureConnections" variable inside the links.js file to **true**. This will affect Web Chat, Co-Browsing and other requests to Oceana.
8. Replace any IP addresses in the links.js file with the FQDN of the server, e.g. the Omni-Channel cluster or the Co-Browsing cluster.

3.5. Server-side configuration for Web Chat

The UI itself does not have any server-side requirements. However, there are various settings for Web Chat that need to be configured in the Omni-Channel Administration Utility (the Admin). To view and configure these settings:

1. Open the Admin.
2. Click "Web Chat" in the bottom left-hand menu. You will see the "Web Chat" menu appear in the top-left panel.
3. To visit transcript configuration, click "Transcripts". You will see the "Transcripts" menu in the top-left panel.

The following subsections detail the Web Chat submenus. Many of these settings will depend upon the attributes for a service; the channel attribute is not included in that.

3.5.1. Config panel

This panel contains the following configuration parameters:

Parameter	Purpose	Value
Keep Alive Time	How long to keep disconnected contacts alive. Not currently used	Minutes / Seconds
Desirable Response	How long to keep disconnected contacts alive. Not currently used	Seconds
Force Idle Customer Check	Check if customers are idle. Current behaviour is to ignore this and take the value of the Force Idle Customer Check Timeout field	true / false
Force Idle Customer Check Timeout	Currently used as a general disconnection timer. If a customer disconnects due to network issues, this value is how long the chat should wait before assuming it is permanent. The reference client's webChatConfig.js file contains a field called "refreshTimeoutSeconds" - that must be updated to match the value of this field.	Seconds
Save Timestamp on Chat Messages	Toggles whether to save the timestamp when persisting the transcript. The default value is true (checked).	true / false
Save Chat History	Toggles whether or not to save the transcript. The default value is true (checked)	true / false
Concurrent Chats Limit per Customer	How many concurrent chats an individual customer may make.	Any number greater than 1
E-mail chat log to Customer	Toggles whether or not to email a transcript to the customer. If this is set to true, then the customer will always receive a transcript, regardless of whether or not they requested one when opening the chat. The default value is false (unchecked)	true / false
Maximum file transfer size	The maximum size in KB of any files that can be transferred from the agent to the customer	Any number greater than 0
External Web Server Domain	Domains that are allowed to open a Web Chat. Enter the hostname/FQDN of your frontend server(s), or set it to the wildcard character (*) to allow chat from all origins	A single hostname or address to limit it to specific servers. Doing so will currently block SMS/Social contacts

3.5.2. Transcripts Config panel (under Transcripts)

This panel configures transcript access for messaging and email contacts. It was previously merged in with the Web Chat config panel. For further details, consult the Transcript Access file.

Parameter	Purpose	Notes
Messaging Transcript Access Web Service	The address of the filtering /transcript access service	The expected URL is http://\${FQDN}:\${port}/CustomerFilterMessages/rest/filter , where \${FQDN} is the fully-qualified domain name or IP address of the server that hosts the CustomerFilterMessages service.
Messaging Transcript Access Send Option	Toggles whether or not to send messaging transcripts to the filtering service.	Options are "Do not send", "Current Transcript" and "Enhanced Transcript". Further details may be found in the "Transcript Access for Messaging and Email contacts" file.
Email Transcript Access Web Service	The address of the filtering service for email	The expected URL is http://\${FQDN}:\${port}/CustomerFilterMessages/rest/filter/transcript/email , where \${FQDN} is the fully-qualified domain name or IP address of the server that hosts the CustomerFilterMessages service.
Email Transcript Access Send Option	Toggles whether or not to send email transcripts to the filtering service	Options are "Send" or "Do Not Send".
Transcript Resend Schedule Number of days to retry	Defines how often the relevant service should attempt to resend failed transcripts.	Transcripts will be marked as permanently failed after this many attempts. For instance, if the value is 10, then the service will attempt to resend failed transcripts 10 times, and then mark them as permanently failed
Transcript Resend Schedule Messaging retry time	Defines the period for when failed Messaging transcripts should be resent	This and the Email retry time should not overlap.
Transcript Resend Schedule Email retry time	Defines the period for when failed Email transcripts should be resent	This and the Messaging retry time should not overlap.

3.5.3. Resources panel

This panel configures welcome messages, the agent label, and the default label for customers.

Parameter	Purpose	Notes
Default Welcome Message	A default welcome message. This will be displayed for any service which does not have a customised message	
Agent Label	Configuring how agents are named in the Web UI	The first option ([Agent]) is the only option where the value can be customised
Customer Label	Configuring how customers are named in the Web UI	This only applies if the customer does not specify their name
Custom Welcome Messages	This sub-panel allows you to define welcome messages for a particular set of attributes	

3.5.4. Auto Phrases

Auto phrases function as groups of suggested phrases for the agent. To add these:

1. Enter the name of the group (e.g. Testing) in the "Name" field, and click the "Create" button. If you already have a group configured, skip this step
2. Select a group from the dropdown menu at the top of the screen.
3. Inside the "Create Auto Phrase" box, enter the distinguishing name of the phrase (e.g. "Hello"), and the text value (e.g "Hi there!"). Click "Create" to add this.
4. Click the left-pointing button (<<) to import a phrase into the group.
5. Click the right-pointing button (>>) to remove a phrase from the group.
6. Under "Edit Attributes", select the attributes that this applies to (e.g. Language.English, Location.Inhouse). It must be an exact match.
7. Click "Save".

3.5.5. Page Push URLs

Page Push URLs function as groups of suggested URLs that agents can push to a customer. To add a new group:

1. Enter the name of the group (e.g. "Testing URLs") in the "Group Name" field, and click the "Create" button. If you already have a group configured, skip this step.
2. Select a group from the dropdown menu at the top of the screen.

3. Enter the URL and a description into the relevant fields, and click the "Create" button. The description is presented to the agent on Workspaces as a reminder of what the URL is for.
4. If the URL is intended to be used for Co-Browsing, click the "Cobrowse" button.
 - a. *When upgrading the Web UI, make sure you upgrade these URLs in particular!*
5. Click "Add" to add a URL into the group.
6. Click "Remove" to remove a URL from the group, and click "Delete" to delete a URL. Deleting is not possible if the URL is in use; it must be removed first.
7. Under "Edit Attributes", select the attributes that this applies to (e.g. Language.English, Location.Inhouse). It must be an exact match.
8. Click "Save".

3.5.6. Web-on-Hold messages

Web-on-Hold messages play in the Web UI while the customer is waiting for an agent, and are subdivided into Web-on-Hold Comfort Groups and Web-on-Hold URLs. To add a new group:

1. Click the relevant tab at the top of the screen (e.g. click "On Hold Comfort Groups" to edit Web-on-Hold Comfort Groups)
2. Click "New" to add new groups, "Edit" to edit an existing one, and "Delete" to delete a group.
3. Fill out the relevant fields, and click "Add" to add the new message/URL to the group.
 - a. The "Delay" and "Hold Time" fields configure the gap between messages.
 - b. The "Description" field in the On-Hold URLs is displayed to the customer, explaining the purpose of the URLs.
 - c. The "Name" or "Tag" fields must be unique, as these are unique identifiers for the group.
4. Click "Remove" to remove a message/URL from the group.
5. Under "Edit Attributes", select the attributes that this applies to (e.g. Language.English, Location.Inhouse). It must be an exact match; if messages are defined for Location.Inhouse, a contact with attributes Location.Inhouse and Language.English will **not** receive those messages.
6. Click "Save".



By design, these are **not** listed in the chat transcript that is sent to the customer. They exist to provide filler on the chat client.

3.5.7. Comfort messages

Comfort messages function as filler while an agent is otherwise engaged during an active chat. They are sent as normal agent messages, and will be present in the chat transcript upon completion of the chat. To add a new group:

1. Click the "New" button.
2. Enter a Name and Delay for the group.
3. Add a Message Text.
4. Click "Add" to add the new message to the group.
5. Under "Edit Attributes", select the attributes that this applies to (e.g. Language.English, Location.Inhouse). It must be an exact match.
6. Click "Save". You must have added at least one message to save the group.

3.5.8. Chat Headers

Chat Headers act as an introduction when sending the chat transcript to a customer via email. There must be at least one set up in advance, with an associated email address. To set up chat headers:

1. Click on "Email" in the bottom-left menu.
2. Click on "Prepared Responses" in the top-left panel.
3. Click on "New Response".
4. Select "Chat History Header" from the "Type" dropdown menu.
5. Enter a name (e.g. "Default Chat History Header") and a subject.
6. Enter the contents of the header into the "Body" box.
7. Click Save.

To set up an email address, refer to the Oceana Deployment guide. When you have a chat history header configured, return to the Web Chat configuration section. To add this into Web Chat:

1. Click on "Chat Headers" in the Web Chat menu.
2. Under "Default Chat Header Settings", set the default header and email address. These will be used as fallback parameters.
3. To assign a header and email address to a particular set of attributes, select the header and address from the dropdown menus at the top of the screen. Choose the required attributes from the "Edit Attributes" box.
4. Click "Save".

4. Upgrading the Web UI

When upgrading the UI, you will need to ensure that any customisations you have made to the code are preserved. The general procedure is as follows:

1. Backup your previous version of the UI, taking note of the URLs and any modifications you have made.
2. Replace the old version with that from the SDK. See the following subsections for advice on specific upgrades.
3. Reapply any modifications that you made.
4. Update the links.js file to match your old links.js
5. Before testing, reset the browser cache and the configuration panel.

More specific procedures are detailed in the following subsections. From 3.4.x onwards, the SDK includes a folder of Upgrade Snippets. These contain text files that detail which lines in the SDK have changed.

4.1. Upgrading from Oceana 3.2.1 to Oceana 3.2.2.x

The 3.2.2.0 and 3.2.2.1 versions of the UI are not backwards compatible with the 3.2.1.0 version (FP1). In particular, the following has changed:

- Calls using the Context Store SDK cannot use a service map. There is a new service, the OceanaCoreDataService, which handles attributes. The full details of changes required to update this are detailed inside the Upgrade_Snippets/Custom_Journey.txt file.
- The URL for the chat WebSocket is now **/services/customer/chat** instead of **/services/websocket/chat**, reflecting the fact that this is no longer used purely for WebSocket redirection. This is a relatively minor change to the links.js file.
- An extra API message for chat has been added to accommodate file transfer messages from an agent. A user interface from Oceana 3.2.1.0 will throw an exception upon receiving this message type, although chat itself will still work. The changes for this are detailed inside the Upgrade_Snippets/Add_File_Transfer.txt file.
- The chat panel does not use the grey sliding tab of FP1. Instead, it uses a jQuery UI dialog widget, following feedback from the DemoAvaya site. The code snippets for this are detailed inside the Upgrade_Snippets/UI_Changes.txt file.
- An alternative method for validating email addresses without using a regular expression has been added to webChatLagon.js. This is not called by default.
- The FP1 UI contained files from CafeX. Due to licencing agreements with CafeX, these must be removed. Delete the **js/libs/la**, **js/libs/cafex** folders.

The following sections detail how to update this.

4.1.1. Adding File Transfer Messages

1. Open the webChat.js and webChatConfig.js files from the FP2 SDK in a text editor.
2. Copy the line **"jsonMethodFileTransfer : 'newAgentFileTransfer',"** into your webChatConfig.js file. Make sure you preserve the comma at the end of the line.
3. Copy the lines **"// placeholder for file transfer notifications. {0} is the file name, {1} is the timestamp. {2} is the URL"** and **"fileTransferMessageText : 'File ({0}) transferred at {1}. Please visit {2} to download it,'"** into your webChatConfig.js file. Make sure you preserve the comma at the end of the second line.
4. Copy the notifyFileTransfer function into your webChat.js file. Make sure you preserve the comma at the end of the line. Note: in 3.2.2.1, this currently needs to be updated to work with a proxy. See the section on passing FileTransferNotifications through a proxy.
5. Copy the handleNotification function into your webChat.js file. Make sure you preserve the comma at the end of the line. This should now resemble the following:

```
/**
 * Handles notification messages.
 *
 * @param message
 */
handleNotification : function(message) { // NOSONAR: too complex
  // for Sonar, but cannot be reduced further
  'use strict';
  var body = message.body, method = body.method;
  if (method === chatConfig.jsonMethodRequestChat) {
    webChat.notifyRequestChat(body);
  } else if (method === chatConfig.jsonMethodRouteCancel) {
    webChat.notifyRouteCancel();
  } else if (method === chatConfig.jsonMethodRequestNewParticipant) {
    webChat.notifyNewParticipant(body);
  } else if (method === chatConfig.jsonMethodRequestIsTyping) {
    webChat.notifyIsTyping(body);
  } else if (method === chatConfig.jsonMethodRequestNewMessage) {
    webChat.notifyNewMessage(body);
  } else if (method === chatConfig.jsonMethodRequestCloseConversation) {
    webChat.notifyCloseConversation();
  } else if (method === chatConfig.jsonMethodRequestParticipantLeave) {
    webChat.notifyParticipantLeave(body);
  } else if (method === chatConfig.jsonMethodRequestNewPushMessage) {
    webChat.notifyNewPagePushMessage(body);
  } else if (method === chatConfig.jsonMethodRequestNewCoBrowseSessionKeyMessage) {
    webChat.notifyNewCoBrowseSessionKeyMessage(body);
  } else if (method === chatConfig.jsonMethodPing) {
    // do nothing with pings. They just confirm that the
    // WebSocket is open.
  } else if (method === chatConfig.jsonMethodFileTransfer) {
    webChat.notifyFileTransfer(body);
  } else {
    throw new TypeError('Received notification with unknown method: ' .concat(method));
  }
}
```

4.1.2. Updating the WebSocket URL

To update the WebSocket URL for chat:

1. Open the links.js file in a text editor.
2. Change the ending of webChatUrl from **services/websocket/chat** to **/services/customer/chat**.

4.1.3. Updating the Customer Journey code

The Customer Journey is now handled mainly by the Oceana Core Data Service. This will provide a consistent schema to preserve routing attributes, thus preventing a customised UI from breaking routing. The Context Store code is still available for non-routing attributes, but any request to Context Store that used a ServiceMap in FP1 must now use the Oceana Core Data Service. To update your UI to include this:

1. Import the following files from the SDK into your Customer Journey pages:
 - a. customerJourneyCommon.js
 - b. oceanaCoreData.js
 - c. contextStore.js
 - d. contextStoreUI.js
 - e. Dependencies for contextStoreUI: these are declared after a comment that says "UI code is declared after bootstrap to fix jQuery conflicts".
2. Copy the URLs for **ocpDataServicesUrl** and **customerManagementUrl** into your links.js file.
3. Replace the AvayaDataStoreClient file(s) with those from the SDK.
4. Replace the videoconferencing page with that from the SDK.
5. Copy the inline setup script from videoconferencing.html to load a customer ID into your customer journey pages.
6. Optional: If the customer's ID is known before they visit the page (i.e. if logging into an account is required), make sure to set the "customerId" variable in sessionStorage, e.g. call **sessionStorage.set("customerId", "100");**
7. Where you previously called **contextStore.addAttribute**, you will now call **oceanaCoreData.addAttribute** if it is a routing attribute. You may still call **contextStore.addAttribute** if the attribute will not be used for routing. The syntax is the same (e.g. **oceanaCoreData.addAttribute("Language.English");** as opposed to **contextStore.addAttribute("Language.English")**)

4.2. Upgrading from Oceana 3.2.2.1 to Oceana 3.3.0.0

There are fewer changes between 3.2.2.1 and 3.3.0.0 than between the 3.2.1.0 and 3.2.2.x versions. Many of these can be integrated by simply replacing the relevant file with that of the SDK.

4.2.1. Logging changes

Log messages were overhauled to be more precise. In addition, they will generally start with any of the following to indicate which area they cover:

- WebChat (e.g. "WebChat: opening the WebSocket")
- EWT (e.g. "EWT: requesting wait time")
- CustomerJourney (e.g. "Customer Journey: creating a context"). This does not affect the contextStore.js file, which is now deprecated.
- CoBrowse (e.g. "Cobrowse: jQuery version")

These can simply be copied from the SDK into your own code, and should not have any affect on functionality.

4.2.2. Changes to the configuration panel

The configuration panel has been updated to allow multiple labs to be configured separately. The purpose of this is to aid development by preventing multiple labs being overwritten. If the webChatConfig.js file has not been edited, then this can be completely replaced with the version in the SDK.

There have also been some changes to the links.js and customerJourneyCommon.js files to go with this. It is recommended that you take note of your current URLs and replace the links.js with that from the SDK, before replacing the URLs with your own. In the customerJourneyCommon.js file, you should change the loadUrlsFromStorage method to resemble the following:

```
*
* Utility to load URLs from local storage while testing. Does not set up chat/EWT/CoBrowsing URLs.
*/
loadUrlsFromStorage: function(){
    'use strict';
    var settings = localStorage.getItem('chatSettings');
    var json = JSON.parse(settings);
    var location = links.getPathNameForConfig();

    if (json !== null){
        var details = json[location];
        var urls = details.urls;
        links.contextStoreHost = urls.contextStoreHost;
        links.secureContextStore = urls.secureContextStore;
        links.oceanaCoreDataServicesUrl = urls.oceanaCoreDataServicesUrl;
    }
}
```

```

        links.customerManagementUrl = 'http://' + links.contextStoreHost + '/services/CustomerManagement';
        if (links.secureContextStore) {
            links.customerManagementUrl = links.customerManagementUrl.replace('http://', 'https://');
        }
    }
}

```

Finally, the certs.html page has been renamed to "temporarilyAcceptLabCerts.html". You can delete certs.html entirely, and replace any reference to it with the new version. **Remove this page in production - it exists purely to work around certificate issues while in a lab enviroment.**

4.2.3. Changes to the Customer Journey

In 3.3, the contextStore.js file is deprecated. Its use remains the same as in 3.2.2.1, but it will be removed in 3.4. The oceanaCoreData.js file has been changed to match an API update in which a touchpoint is now referred to as "journeyElement"; the JavaScript itself has not changed, but the URLs that are constructed for the request must be updated. For example, the updateAttributes function should create a URL for the request as shown below. Apply this to any method which includes "?touchpoint=" in a URL.

```

var URL = links.oceanaCoreDataServicesUrl + 'update/context/' + customerJourneyCommon.contextId + '?
journeyElement=' + customerJourneyCommon.pageTouchpoint;

```

The createContext method will now specify that attributes will be persisted to the External Data Mart (EDM), allowing these attributes to be displayed on Workspaces as part of the customer journey. This fixes a bug that was present in 3.2.2.1. If your copy of oceanaCoreData.js does not include this, update the createContext method to resemble the following:

```

/**
 * Create a new Context using the Oceana Core Data Services.
 */
createContext : function() {
    'use strict';
    if (oceanaCoreData.csInstance === null || avayaGlobal.isEmpty(oceanaCoreData.attributes) ||
        oceanaCoreData.customerId === '') {
        avayaGlobal.log.warn('CustomerJourney: Cannot create context if customerID or attributes are
empty! Request a customer ID and add attributes first');
        return;
    }

    // create the schema. Creating context requires a group ID, a context ID and whether or not to persist
to the EDM,
    // but updating doesn't, so add these here.
    avayaGlobal.log.info('CustomerJourney: Creating a context');
    var json = oceanaCoreData.createSchema();
    json.groupId = customerJourneyCommon.customerId;
    json.contextId = customerJourneyCommon.contextId;
    json.persistToEDM = true;
    var contextRequest = new XMLHttpRequest();
    contextRequest.timeout = 10000;
    contextRequest.open('POST', links.oceanaCoreDataServicesUrl + 'context/schema?journeyElement=' +
        customerJourneyCommon.pageTouchpoint);
    contextRequest.setRequestHeader('Content-Type', 'application/json');
    contextRequest.addEventListener('readystatechange', oceanaCoreData.handleContextResponse);
    contextRequest.send(JSON.stringify(json));
},

```

4.2.4. Opening a chat

In 3.2.2.1, the webChatSocket.js file opened a chat using four parameters (username, email address, account number and phone number). The account parameter was not used anywhere, so it was removed from the chatLogin method inside webChat.js as part of a code cleanup effort for 3.3.0.0. This method is called inside the handleOpen method of webChatSocket.js; the following code snippet illustrates the difference.

```

webChat.chatLogin(webChat.g_user, webChat.g_email, webChat.g_account, webChat.g_phone);
webChat.chatLogin(webChat.g_user, webChat.g_email, webChat.g_phone);

```

4.2.5. Noscript headers added to home.html page

The home.html page now includes two div elements at the top of the page that are normally hidden on page load. However, users who block JavaScript (e.g. by using the NoScript addon for Firefox) will see the following sentences:

- "If you see this, then either you have disabled JavaScript, or there is a syntax error in the Avaya WebChat files"
- "If you see this, then either you have disabled JavaScript, or there is a syntax error in the Avaya CoBrowsing files."

The purpose of these divs is to allow security or privacy-minded users who block JavaScript know that this is required to use WebChat or Co-Browsing; they will also appear if there is a syntax error in the code, allowing them to serve as a quick sanity check. To use these in your website:

1. Copy the divs from home.html into your own home.html page.
2. Add the following lines into the initialisation functions at the bottom of webChatUI.js and coBrowseUI.js, respectively.

```
$('#chatJavaScriptAlertHeader').hide();
$('#cobrowseJavaScriptAlertHeader').hide();
```

There is currently no equivalent header for the customer journey pages. To add these:

1. Open the products.html page.
2. Inside the **<body>** tag, add a div element with an id of your choice, e.g. "customerJourneyJavaScriptAlertHeader".
3. Set the text content of the div to a message of your choice, e.g. "Avaya Web Chat requires JavaScript to function. Please enable JavaScript to use this page. If you are using NoScript, you will also need to allow JavaScript from a.b.c.d", where a.b.c.d is the hostname or IP address of the cluster that hosts ContextStore, the CustomerManagement service and the Oceana Core Data service.
4. Inside contextStoreUI.js, add a line similar to the above to the "setupUI" function. It should resemble the following:

```
/**
 * Set up the UI properly. Uses a zero-millisecond timeout to "pause" it until the rendering has finished.
 */
var setupUI = function() {
    setTimeout(function() {
        $j('.button').button();
        $j('#customerJourneyJavaScriptAlertHeader').hide();
    }, 0);
};
```

4.3. Upgrading from 3.3.0.0 to 3.4.0.0

4.3.1. Converting links.js to use hostnames instead of full addresses

To go with the deployment simplification feature in Oceana 3.4.0.0, the Web UI was converted to use hostnames/IP addresses in the links.js file, instead of the full URL. Refer to the Upgrading_Links.txt file in the Upgrade Snippets folder.

4.3.2. Fixing the routePointIdentifier field when requesting a chat

When requesting a chat, there is an optional field for the routePointIdentifier. However, before 3.3.0.1, this was mistakenly sent up as "routePointIdentifier". To fix this, rename the relevant field inside the chatLogin function of webChat.js to "routePointIdentifier".

4.3.3. Estimated Wait Time changes

In Oceana 3.4.0.0, the Web UI implementation for Estimated Wait Time (EWT) has been changed to account for the number of logged in agents. It will also assume that the chat is *not* available if the request returns any kind of error. To import these changes:

1. Open the webChat.js file.
2. Change the **maxWaitTime** variable to 600. This is required, as the value for EWT that is returned by WorkAssignment is returned in seconds.
3. Copy the contents of the **parseServiceMap** function, and paste it into your copy.
4. Copy the **addEwtToChatTab** function from webChatUI.js into your own copy. This provides an alternative to displaying an alert box, and is optional. If you do not wish to use this method, remove the line which calls this in the **parseServiceMap** function.
5. Copy the updated log statements into your webChat.js file. This step is optional, but highly recommended.

The criteria for offering chat are as follows:

- If the request does not return any metrics, chat is assumed to be available.
- If the estimated wait time is between 0 (inclusive) and 600 seconds, and there are agents logged in for the requested service, chat is available.
- The chatbot does not count as an agent for the purposes of estimating wait time. If no agents are logged in, then chat is not available, regardless of whether the chatbot is installed and configured.

4.3.4. Error handling improvements

The Web UI's error handling was improved for 3.4.0.0. These changes are recommended, but not strictly required. To copy these changes:

1. Copy the **parseErrorMessage** function from webChatSocket.js into your own copy.
2. Update the **handleMessage** function in the same file to include a call to **parseErrorMessage**, as in the SDK.
3. Copy and replace any line that uses the logger (avayaGlobal.logger).

4.3.5. Customer Journey changes

The contextStore.js file, which was deprecated in 3.3.0.0, has now been removed. In addition, the oceanaCoreData.js file has been updated to allow more options. As the API for this file has not changed, it can simply be replaced. If you wish to add more data about the user, call the **addDataPair** function inside oceanaCoreData.js.

4.3.6. Initialisation script changes

In previous releases, there were initialisation methods scattered throughout the JavaScript files. These were bundled with but separate from the modules themselves. To simplify deployments, these were refactored into the modules. The snippets are listed inside the Refactoring_Init_34.txt file inside the Upgrade_Snippets folder.

4.4. Upgrading from 3.4.0.0 to 3.4.0.1

In 3.4.0.1, the following issues were fixed:

- After a chat ended, the chat panel was supposed to close immediately if the customer ended the chat, but not if an agent closed it. Subsequent chats would close the panel immediately, regardless of who ended it.
- Duplicate notifications that an agent had left the chat. By design, Oceana sends multiple participant leave notifications if an agent leaves a chat during a supervisor observe/coach/barge session.
- The hidden address field in the chat login panel was set to an empty string by default.
- The configuration panel now includes options to configure reconnection settings (frequency and number of attempts to make).

These are documented in the Upgrade_Snippets/3.4.0.1/3.4.0.1_Fixes.txt file.

4.5. Upgrading from 3.4.x to 3.5.0.0

4.5.1. jQuery and Bootstrap updates

The jQuery, jQuery UI and Bootstrap versions have been updated to use the current latest (3.3.1, 1.12.1 and 4.0.0, respectively). The scripts will now be loaded via a content distribution network (CDN) for performance reasons, and have been moved around in the SDK folder. Refer to the "jQuery_Updates.txt" file inside the Upgrade_Snippets/3.5.0.0 folder.

4.5.2. Extra page for demos

There is a new page for demos, demo.html. This is a stripped-down version of the home page, which includes an iframe that is used to include a customer's site for demo purposes, and the ability to load the SMSVendorSnapiin for testing SMS contacts. To copy this into your version:

1. Copy the demo.html page into your website directory.
2. Copy the smsVendorLinkElements folder into your website.
3. Copy the demoPage.css file into your website. This contains the styling for the demo.html page.

Remove the demo.html page in production - it exists purely for demos.

4.5.3. New stylesheets and files for configuration panel

The configuration panel is managed by a separate JavaScript and CSS files, chatConfigPanel.js and chatConfigPanel.css. The purpose of these is to make it easier to convert a demo page to production. To convert this:

1. Replace your copy of webChatConfig.js with the version from the SDK.
2. Copy chatConfigPanel.js into your chat pages.
3. Copy the chatConfigPanel.css file into your chat pages.

The chat elements are now styled by a standalone CSS file, chat.css. Add this into your chat pages.

4.5.4. General code cleanup

Some sections of the code were cleaned up. Refer to the "Code_Cleanup.txt" file.

4.5.5. Email validation changed

The email validation in the Web UI was becoming difficult to maintain and essentially duplicated work on the CustomerControllerService. It was planned to be stripped down in 3.4.0.1, but did not make it in. To use this, consult the "3.4.0.1_Fixes.txt" file in the 3.4.0.1 folder.

4.5.6. Watson Integration

This was added as a patch for 3.4.0.1, referred to here as 3.4.0.1.20. To add this into your Web UI:

1. Open the webChat.js file from the SDK.
2. Copy the following methods from the webChat.js file into your version:
 - a. chatLogin
 - b. notifyNewMessage
 - c. sendChatMessage
 - d. sendWidgetMessage
3. Copy the "customData" object into your webChat.js file.
4. Open the webChatSocket.js file in the SDK.
5. Copy the sendMessage function and the line that declares the authentication token into your copy. You may leave the authentication token as a blank string.

4.5.7. Reloading chat after page refreshes

This was added along with the Watson Integration feature. The purpose is to allow the customer to refresh the page or visit another chat page without ending the chat. Refer to the "Reload after Refresh" file inside the 3.5.0.0 folder.

There are two caveats to beware of:

- If the customer does not connect within 30 seconds, the Oceana Chat service will assume that the customer has left permanently.
- There is a method to rebuild the chat messages from the UI's perspective upon reconnection. However, this is not called out-of-the-box. To fix this, call the loadTranscript function inside webChatSocket.js when the chat has been opened. This will be fixed in a future release.

4.5.8. Customer Management updates

As part of a request to standardise customer details, the Web UI login panel was changed as follows:

- The customer's name is now split into separate fields for their first and last name.
- The phone number is split into 3 fields: one for the country code, one for the area code, and one for the number.
- The ability to add a custom field for demo purposes.
- A topic field was added below the custom fields entry. Note: In 3.5.0.0, this is currently labelled as "Subject", but should be labelled "Topic"; this will be fixed in a future release. The field does not include any tooltip to explain invalid characters; this was a design decision to unblock delivery of the core work.

Refer to the "Customer Management Updates" file inside the 3.5.0.0 folder.

4.6. Upgrading from 3.5.0.0 to 3.5.0.1

The 3.5.0.1 updates are mainly bug fixes. The following issues were fixed:

- The page refresh mechanism did not account for the disconnection timeout on the server. The fixes for this are documented inside the "Fixing_Page_Refresh" file.
- Due to an API mismatch in the CustomerControllerService, the "topic" field would not be received when opening a chat. The Web UI itself worked as expected, but there are some naming issues that should also be fixed. These are documented inside the "Page_Topic" file.
- A workaround for a Chrome bug which causes Chrome to report WebSocket connections as closing abnormally, when connecting to a two-node cluster. The fixes for this are documented inside the "Fixing_Reconnection_Chrome" file.
- The customData object was undefined by default when opening a chat. This is listed inside the "Miscellaneous" file.
- A Strategy field for routing is required when creating the context, but this is not currently validated by Oceana. The Web UI fix is listed inside the "Miscellaneous" file.
- The "loadTranscript" function inside webChatSocket.js is now called when reloading the page while a chat is in process. Refer to the "Miscellaneous" file.
- The maximum length of various fields were updated to align with the CustomerController WebSocket endpoint. Refer to the "Miscellaneous" file.

These files can be found inside the 3.5.0.1 folder.

4.7. Upgrading from 3.5.x to 3.6.0.0

These files can be found inside the 3.6.0.0 folder.

4.7.1. Removing AvayaClientServices file

The AvayaClientServices file was removed due to being effectively unused. As a result of this, the UI only uses the standard browser console for logging. Refer to the "Removing AvayaClientServices" file.

4.7.2. Refactoring DOM interactions to a common file

In previous versions of the Web UI, interactions with the page were scattered throughout the various files. For the sake of consistency and to aid development, these have all been moved into webChatUI.js and contextStoreUI.js. The changes are listed in the "Refactoring DOM Interactions" text file.

4.7.3. Miscellaneous fixes

Aside from general code cleanup and similar fixes (documented inside the Miscellaneous.txt file), the following has changed:

- The configuration panel will now display the development sprint number. Since the configuration panel is meant to be simply replaced and only used inside a lab environment, there are no upgrade snippets for this.

- The AvayaCoBrowsingClientServices library has been updated to the 3.6.0.0 release. The upgrade procedure is to simply replace the contents of the cobrowse-oceana folder.
- A new CSS class was added to serve as a selector for URLs that should not be clicked during a CoBrowsing session. The fixes are documented inside the Miscellaneous.txt file.

4.8. Upgrading from 3.6.0.0 to 3.6.1.0

These files can be found inside the 3.6.1.0 folder. Most of these are bug fixes, and will be listed inside the [Miscellaneous.txt](#) file. The following are more specific:

4.8.1. Refactoring how customer details are stored during a chat session

The customer details that are gathered when opening a chat are now refactored into a single object for convenience. The exact syntax is shown below:

```
{
  "firstName": "M",
  "lastName": "Smith",
  "email": "msmith@test.com",
  "phone": {
    "country": "+555",
    "area": "111",
    "number": "23456"
  }
}
```

The upgrade instructions are listed inside the [Refactoring_Customer_Details.txt](#) file

4.8.2. Refactoring the customer journey code

The customer journey code has been simplified to:

- Merge customerJourneyCommon and oceanaCoreData into one file. The split was a legacy of an earlier release where the primary customer journey service was being migrated from Context Store to the Oceana Core Data Service; during the transition period, the Web UI had to support both.
- Setting the "touchpoint" or journeyElement is no longer supported. The ability to do so was mistaken.
- Adding attributes does not automatically create or update the context in Oceana Core Data Service. There are methods to do so in the API, and the customerJourneyUI.js file contains methods to invoke these.

The instructions and expected flow are listed inside [Refactoring_Customer_Journey.txt](#)

4.8.3. Refactoring system messages

In previous releases, system messages (e.g. "Sending Login Details" when opening the chat or "Opening chat after page refresh" when reloading the page during an active chat session) were scattered throughout the various Web Chat files. These have been refactored into the webChatConfig.js file for ease of customisation and localisation. Refer to the [Refactoring_System_Messages.txt](#) file.

4.8.4. File Transfer Notifications no longer use a direct download link

The file transfer notifications no longer use a direct download link. Instead, the client builds the URL using two specific parameters, making it easier to pass through a reverse proxy. Refer to the FileTransferFixes.txt file.

4.9. Upgrading from 3.6.x to 3.7.0.0

These can all be found in the 3.7.0.0 folder. Aside from miscellaneous fixes in the [Miscellaneous.txt](#) file, here are the more specific features that were added.

4.9.1. Refactoring page refresh mechanism

In previous releases, the client kept a local copy of the transcript to rebuild the chat after refreshing the page. This was moved into the CustomerController. The fixes for this are documented in the [Page_Refresh_Updates.txt](#) file

4.9.2. Support for transfer-to-user feature

The fixes for this involve adding an extra possible reason for why an agent has left the chat, and are documented inside the [Transfer_To_User.txt](#) file.

4.9.3. RTL support

RTL support consists of setting the page direction so that it is read from right-to-left. The fixes for this can be found in the [Page_Direction.txt](#) file.

4.10. Upgrading from 3.7.0.0 to 3.7.0.1

A new CoBrowse SDK has been bundled with the artifact.

The page refresh mechanism was refactored to take an optional parameter, *requestFullTranscript*, that will configure whether or not the CustomerController will send the entire transcript or just any messages that were missed during a reconnection. The *Page_Refresh_Updates.txt* file has been updated to account for it.

4.11. Upgrading from 3.7.x to 3.8

These can be found in the 3.8.0.0 folder.

4.11.1. Max length of a text message reduced

The stated max length of 10000 characters for a chat message was actually not achievable due to Jetty limitations. Since 10000 characters was deemed to be a bit excessive, this has been reduced to 2500 characters. The fixes for this are documented in the *Max_Text_Length.txt* file.

4.11.2. Topic field was improperly validated

The topic field in the opening chat panel did not correctly align with the internal API. The fixes for this are included in the *Sanitisation_Changes.txt* file.

4.11.3. Internal Oceana changes

The CustomerControllerService in Oceana will now remove invalid attributes when opening a chat. The list of valid attributes are read from the database at 5-second intervals. This does not directly affect the client.

4.12. Upgrading from 3.8.0.0 to 3.8.0.1

There is one change to the sample client but not the API, which has been added to the *Select_Showing_Observing_Supervisor* file the 3.8.0.0 folder: configuring whether or not to actually show that an observing supervisor is present in the chat. The chatConfigPanel.js file contains additional methods to configure it, and there are further HTML elements added to the home.html page to use these, but these are not required for production use.

4.13. Upgrading to 3.8.1

A few issues have been fixed. All of these are with the sample client and are documented in the files inside the 3.8.1.0 folder:

- The sample application did not properly parse widget messages from automated agents (bots), effectively hiding them from the user.
- The labels used for distinguishing bot messages from live agent messages was inconsistent.
- There was a minor CSS issue where text from the bot is always oriented left-to-right.

5. Customising the Web UI

The Web UI is a reference application to demonstrate the functionality of Oceana, and will need to be edited to suit your organisation. This requires some knowledge of HTML, JavaScript and CSS.

5.1. How to edit the UI

The Web UI files may be edited using any text editor or IDE (integrated development environment) that supports JavaScript, HTML and CSS. The following are examples which are known to work:

- Eclipse (IDE)
- NetBeans (IDE)
- Notepad++ (text editor)
- Notepad (text editor)
- Gedit (text editor)
- Vim (console-based text editor)

Most of the customisation is done inside the stylesheets, which are located under the *src/main/webapp/css* folder. This contains the following files:

- *style.css* - this is the main stylesheet.
- *chat.css* - this is a stylesheet used exclusively by the chat components (e.g. the chat panel).
- *configPanel.css* - this is a stylesheet used exclusively by the chat configuration panel. This can be safely removed in production environments.
- *cobrowse.css* - this is a stylesheet used exclusively by the cobrowse.html page, which is used to demonstrate Co-Browsing.
- *certs.css* - this is a stylesheet used exclusively by the temporarilyAcceptLabCerts.html page, which is used to help setup a demo environment. You can remove this for a production site.

Other third-party stylesheets are located under *src/main/webapp/assets/css*. This includes one for Bootstrap, which has been used to create a mobile-friendly version of the UI.

The functionality of the JavaScript library is done in the following files, which are located inside the *src/main/webapp/js* folder:

File	Purpose
avayaGlobal.js	Holds utility functions, such as finding DOM elements or validating strings/numbers. Add any common functions you require to this file, and include it on all pages that have some interaction with Oceana (chat, adding attributes, etc.).
chatConfigPanel.js	Handles the configuration panel functionality for demos and testing. This file should be removed in a production environment. When updating the UI, you can simply replace this file entirely.

js	
customerJourney.js	Handles the customer journey interactions with CustomerManagement and OceanaCoreDataService.
customerJourneyUI.js	Handles the UI interactions for the customer journey.
estimatedWaitTime.js	Handles estimated wait time (EWT).
links.js	Configures URLs for the chat and ReST requests. Configure this as part of the deployment process.
webChat.js	Defines the chat functionality. Change variables such as routePointIdentifier and workflowType to configure the chat behaviour.
webChatConfig.js	Configures timers, flags and other variables for the chat. Change the timers and flags to suit your preferences. In Oceana 3.4.0.1 and earlier, this also contains methods to configure chat inside a lab environment, and will save these to localStorage; in 3.5.0.0, those were refactored into chatConfigPanel.js
webChatLogon.js	Handles the logon features. In particular, it gathers customer details from the login panel and performs some basic validation.
webChatSocket.js	Handles low-level WebSocket interactions such as sending or receiving messages.
webChatUI.js	Contains code to handle UI interactions for the chat. It uses jQuery UI methods by default; replace these if you do not wish to use this.

The files required for Co-Browsing are located inside the **src/main/webapp/cobrowse-oceana** folder. This is divided into the following sub-folders:

- css - holds the required Cascading Style Sheets for the Oceana Co-Browsing components.
- js - holds the required JavaScript files for the Oceana Co-Browsing component. This folder also contains required dependencies in the libs folder.

Do not change the contents of the files in any of the following folders, as they contain required dependencies for Co-Browsing or other components:

- src/main/webapp/js/libs
- src/main/webapp/cobrowse-oceana/js/libs

The HTML pages cover different scenarios, and are listed here for convenience:

- home.html - the main page. This contains a chat panel.
- demo.html - a secondary chat page. This contains an iframe linking to Wikipedia; the idea here is to allow the contents of this iframe to be replaced with a customer site for demos.
- cobrowse.html - a sample page to demonstrate Co-Browsing. Do not add the chat panel or Customer Journey code here.
- thankYou.html - a sample page to demonstrate swapping pages during a Co-Browsing session. Do not add the chat panel or Customer Journey code here.
- products.html - a sample page for available products. Do not add the chat panel or webChat JavaScript files to this page.
- services.html - a sample page for available services. Do not add the chat panel or webChat JavaScript files to this page.
- solutions.html - a sample page for available software solutions. Do not add the chat panel or webChat JavaScript files to this page.
- videoconferencing.html - a sample page for video-conferencing equipment. Do not add the chat panel or webChat JavaScript files to this page. This page is used to start the Customer Journey.

5.2. Opening and closing a chat

The default way to open a chat is to use the following functions in the following order:

```
// Gather customer details first and set the webChat.custDetails object first
// This is effectively what happens inside the logon() method inside webChatLogon.js
var custDetails = {
  "firstName" : "J.",
  "lastName" : "Smith",
  "email" : "smithj@example.com",
  "phone" : {
    "country" : "+353",
    "area" : "01",
    "number" : "1234567890"
  }
}
```

```

};
webChat.custDetails = custDetails;

// Get the endpoint for the chat.
// The sample app uses a global WebSocket object due to legacy reasons.
var url = links.getWebChatUrl();
websocket = new WebSocket(url);

// define event handlers. These are contained in the webChatSocket.js file
// the onClose method will be shown here in more detail
websocket.onopen = chatSocket.handleOpen;
websocket.onmessage = chatSocket.handleMessage;
websocket.onerror = chatSocket.handleError;
websocket.onclose = function (event) {

    // If it is an expected/graceful close, do not attempt to reconnect.
    // The flags here are required due to an issue with Chrome's engine which results in it (or any browser
based upon it)
    // mistakenly thinking that the connection closed with status code 1006 (abnormal closure) if pointing
at a multinode Omnichannel cluster
    // Status codes 1000 and 1005 are considered to be graceful closures
    if (!chatConfig.previouslyConnected || chatConfig.dontRetryConnection || event.code === 1000 || event.code
=== 1005) {
        avayaGlobal.clearSessionStorage();
        chatSocket.handleClose(event);
    } else if (event.code === 1001) {
        // this will occur if the customer leaves or reloads the page.
        // The purpose here is to allow the user to continue a chat while browsing other pages
        chatSocket.setupRefresh();
    } else {
        chatSocket.reconnect();
    }
};

// send a request to start the chat. This is internally called by the default handleOpen function.
webChat.chatLogin();

```

Closing a chat can be done in any of the following ways:

```

// This flag MUST be set to avoid Chrome mistakenly believing that the connection closes abnormally,
// if you have connected to a multinode Omnichannel cluster.
// Not doing so will result in the customer attempting to reconnect to a session that has been cleared!
chatConfig.dontRetryConnection = true;
websocket.close();

// If you wish to simulate an invalid close attempt and reconnection, you can call either of the following
lines.
// The code here can be 3000 or above, as these are not reserved for WebSocket extensions
// The second line contains a close reason - this is entirely optional, as per the WebSocket spec
websocket.close(3000);
websocket.close(3000, "Testing reconnection");

// The difference between this and the above is that this will wait for Oceana to send back a
CloseConversationNotification before invoking the above lines.
webChat.quitChat();

```

5.2.1. Close Reasons

Close reasons are an optional part of the WebSocket specification. Oceana will internally log them and convert to a range of possible values, reproduced below. Some of these (underlined and in italics) will not be sent to the client.

- UNKNOWN. This is a fallback value for any possible String that does not fit into any of the following categories. In practice, when this occurs Oceana assumes that the client has left
- REASON_SESSION_RECONNECT. This will occur when reconnecting a session, and will result in the old endpoint closing. This will not be shown to the client.
- REASON_MAINTENANCE. Oceana is shutting down for maintenance

- **REASON_CLOSING_CLEANUP.** This is always the last step in closing. It may be sent to the client if the administrator has changed the "Close all Chatrooms" attribute in System Manager to *true*.
- **REASON_IDLE.** Oceana has decided that the connection has been idle for too long, and closes the connection. This is most likely to occur if the client connection reconnects without actually sending a RenewChatRequest.
- **REASON_TIMED_OUT.** The customer refreshes or leaves the page and fails to reconnect within a certain time period. Simply reconnecting the WebSocket is not enough to resume the chat; send a RenewChatRequest to let Oceana know that the client has reconnected.
- **REASON_FAILED_OPEN.** This will be logged and sent to the client if an error occurs when opening the session. This is invariably due to an internal problem with routing or creating a contact.
- **REASON_AUDIT.** An internal audit mechanism is closing the session to avoid an inconsistent state. Will only appear in the Oceana logs for the CustomerControllerService.
- **REASON_CLOSE_ADMIN.** An administrator manually closed the chat using the Oceana Data Viewer. If the chat is in progress, this may be sent down to the client.
- **REASON_NODE_SWAP.** This occurs when swapping nodes after a reconnection, and will simply close the old WebSocket endpoint.
- **REASON_CUSTOMER_LEAVE.** The customer has decided to close, and is the default value.
- **REASON_STOPPING_SMS_SOCIAL_SERVICE.** This is sent by the MessagingService when stopping inbound SMS/Social contacts, and has no impact on chat.

As per the WebSocket specification, these can be at most 123 bytes.

5.3. Which methods to call for chat

The following methods can be found inside the webChat.js file. The WebSocket must be opened first; this is performed by the **openSocket** method inside webChatSocket.js.

Request method name	Purpose	Notes
chatLogin	Send the NewChatRequest or RenewChatRequest object to start /reconnect the chat	
sendChatMessage	Send a NewMessageRequest for a regular chat message	
sendWidgetMessage	Send a NewMessageRequest for a widget message (related to Watson integration)	
sendsTyping	Send an IsTypingRequest to let the agent know that the customer is typing	
sendCoBrowseStatus	Send an CoBrowseStatusRequest to let the server know if Co-Browsing is enabled on this page	
sendPing	Send a Ping message to the server to check if the connection is still alive (because JavaScript doesn't <i>have</i> a builtin method)	The server will send back a Pong, but the UI does nothing with this
quitChat	Send a CloseConversationRequest to the server	
Notification method name	Purpose	Notes
notifyRequestChat	Let the user know that they have entered the chat	
notifyRouteCancel	Let the user know that routing has been cancelled (e.g. no agents available)	
notifyNewParticipant	Let the user know that a new participant has joined	
noytifyParticipantLeave	Let the user know that a participant has left	May result in the conversation closing, if the agent has not transferred to another service
notifyNewMessage	Let the user know that the agent has sent a message	Can be a regular chat message, or contain a widget
notifyIsTyping	Let the user know that an agent is typing	
notifyCloseConversation	Let the user know that the conversation is closed	
notifyNewPagePush	Let the user know that the agent has pushed a URL	May be used to open a Co-Browsing session
notifyNewCoBrowseSessionKeyMessage	Let the user know that the agent has initiated a Co-Browsing session	
notifyFileTransfer	Let the user know that the agent has sent them a file	Prior to 3.6.1, this included a direct download link. In 3.7, that has been removed entirely

5.4. Configuring the appearance of the chat panel

The appearance of the chat panel is largely configured inside the chat.css file. Changing this requires some knowledge of Cascading Style Sheets. Some of the most common changes are:

- To change the background of the chat panel, open the chat.css file and search for the **#chatPanel** entry. Add an entry for "background" with the colour of your choice (e.g. `background: #222222`), and add an entry for text colour so that text stands out against the background (e.g. `color: #ffffff`).
- To change the text for messages from an agent, change the contents of the **p.response** entry. The default values (`color: black; font-style: italic; text-align: right`) result in messages from the agent appearing to be italicised in black, and aligned along the right-hand-side of the transcript window.
- To change the text for messages from a customer to an agent, change the contents of the **p.sent** entry. The default value (`color: red`) results in messages from the customer appearing to be red. By default, these will be appended on the left-hand-side of the transcript window.
- System messages (e.g. notifications that an agent has arrived or left) are by default left-aligned in blue italics. To change this, change the **"p.system"** entry.
- Chatbot messages are by default shown in a different style to agent message. To configure this, change the **"p.chatbot"** entry. If you wish to treat these as normal agent messages, either replace the contents of the `p.chatbot` entry with that of `p.response`, or update the `notifyNewMessage` method in `webChat.js` to treat the bot as another agent.
- Dates and names are displayed above their respective sender, e.g. messages from the customer will be displayed with their name and the timestamp above them. There are two classes for this: the **"p.date"** and **"p.agentDate"** classes. The former is used to display the customer's name and timestamp, while the `agentDate` class is used to display the agent's name and timestamp.

The styling used by the buttons is defined by the minified CSS files inside the `jquery-ui` folder. If this style does not match your preferences, there are several ways you can change this:

- Add a style attribute to each button you wish to change. If you only have one or two buttons to change, this is a relatively quick and easy process, and may be performed as a first step to change the appearance before following one of the methods below. An example of this is:

```
<button class="button" style="background:green;">Click me</button>
```

- Create a new CSS class and assign that to each element you wish to change. If you have multiple elements that need changing, this is a better method. An example of this is:

```
<button class="myNewButton">Click me</button>
```

- Visit <https://jqueryui.com/> and click the "Custom Download" button. Scroll down to the "Theme" section, and click the URL for "design a custom theme". This will allow you to create a customised theme for the Web UI.

5.5. Setting custom fields

Custom fields are customised key-value pairs that are stored in the Omnichannel Database. To add them, call the `webChat.addCustomFields` method before opening the chat. The API has the following requirements:

- The key/title of the pair must be unique, must not be blank, and must be at most 50 characters.
- The value of the pair may be blank, but must not exceed 255 characters.

5.6. Passing FileTransferNotifications through a proxy (Oceana 3.2.2.1 and earlier)

The URL in the `FileTransferNotification` message currently is a direct download link to the `CustomerControllerService`. If you are using a proxy to keep Oceana isolated from the public Internet, this may cause `FileTransferNotifications` to fail. To fix this, open the `webChat.js` file and locate the `notifyFileTransfer` method. Change the contents of this method to resemble the following:

```
'use strict';
avayaGlobal.log.info('WebChat: Notifying of file transfer');
var url = body.address;
var filename = body.name;
var timestamp = new Date().toLocaleString();
var message = chatConfig.fileTransferMessageText;

// account for proxies.
url = url.replace('ccHost', 'proxyHost');

message = message.replace('{0}', filename);
message = message.replace('{1}', timestamp);
message = message.replace('{2}', url);
webChat.writeResponse(message, chatConfig.writeResponseClassResponse);
```

Replace `ccHost` and `proxyHost` with the hostname of your Omnichannel node and proxy, respectively. This does not apply to Oceana 3.3.0.0 and later; this fix is already present there. In 3.6.1 and later, the client will build the URL from scratch using the file UUID and the contact's work request ID.

5.7. Editing or suppressing agent presence notifications

By default, the Web UI will display a message to announce that an agent has joined or left the chat. To change this functionality, edit the **notifyParticipantJoin** and **notifyParticipantLeave** functions inside `webChat.js`. To suppress the alert entirely, remove the `"webChat.writeResponse"` line from both functions.

The `webChatConfig.js` file includes a flag, **suppressChatbotNotifications**, which can be used to hide announcements for the Automated Chat Agent. By default, this is set to `false`, which results in the UI announcing the arrival or departure of the Automated Chat Agent. Setting this to `true` will result in the arrival or departure of the Automated Chat Agent not being displayed to the customer.

To suppress notifications of a supervisor observing the chat, coaching the agent or barging into the room, edit the **notifyOfObserve**, **notifyOfCoach** or **notifyOfBarge** flags in the `webChatConfig.js` file. By default, these are set to `true`.

5.8. Displaying Co-Browsing URLs in the chat transcript

The Web UI will display a message to announce the arrival of a Page Push Message, and then display the URL separately. Co-Browsing URLs will be announced using a separate message, and by default will not display the URL in the transcript. To change this behaviour, locate the `notifyPagePushMessage` function inside the `webChat.js` file, and move the following line outside the `else` block containing it:

```
webChat.appendLink(url, destination, chatConfig.autoOpen, webChat.messages);
```

To make the Web UI treat Co-Browsing URLs as Page Push Messages, remove the `if/else` block entirely.

5.9. Adding or removing log messages

Adding or removing log statements to a method has no impact on the code, but may aid in debugging or troubleshooting. In Oceana 3.6 and later, this uses the standard browser console. Prior to this, logging was one by any of the following commands:

- `avayaGlobal.log.info("foo")` - creates an info-level message that logs "foo" to the console. This is distinguished by the info symbol in the message, and should be reserved for high-level information.
- `avayaGlobal.log.debug("bar")` - creates a debug-level message that logs "bar". This is visually indistinguishable from the normal `console.log("bar")` command in the console.
- `avayaGlobal.log.warn("foobar")` - creates a warning message that logs "foobar". This is distinguished by the yellow triangle and yellow background. This should be used for events that may cause a problem, but do not break functionality.
- `avayaGlobal.log.error("Undefined object")` - creates an error message. This is distinguished by an error symbol in the message, and a red background. This should be used for errors.
- `avayaGlobal.log.fatal("Out of memory")` - creates a fatal error message. This is marked by the word "fatal" prefacing the message (e.g. this example would log "fatal Out of memory"). This is not used anywhere, and should be used for serious errors only.
- `avayaGlobal.log.trace("Examining nested function")` - creates a trace-level message. This dumps a stack trace onto the console, so only use this for debugging.

By convention, the log messages in the client will start with the following text to indicate what area they affect. Try to follow this convention when adding your own logging messages.

- `WebChat` - covers log messages exclusive to WebChat.
- `WebChat/CoBrowse` - covers messages that involve WebChat and Co-Browsing. This mainly occurs when an agent starts a Co-Browsing session during a WebChat session.
- `CoBrowse` - covers Co-Browsing events.
- `EWT` - covers the estimated wait time events.
- `Customer Journey` - covers the customer journey and context gathering.

5.10. Adding chat attributes

The attributes used when opening a chat are defined in the `webChatLogon.js` file. This takes an array of String values, e.g. `["Location.Inhouse", "Language.English"]`. Add or remove values from this to change the default attributes for a chat. The `webChatLogon.js` file also includes API functions for adding attributes to or removing them from this array. These are currently used in the configuration panel. Use the following commands to add or remove Product Keyboard:

- `chatLogon.addAttribute("Product.Keyboard");`
- `chatLogon.removeAttribute("Product.Keyboard");`



Note: The attributes here are separate from the Customer Journey attributes. However, they share the same system-wide limit of 10 attributes. The Web UI will not allow more than 10 attributes to be shared between Web Chat and Customer Journey; this is due to an API limitation in the WorkAssignment service.

5.11. Changing the Estimated Wait Time request and response

The `estimatedWaitTime.js` file includes methods to request the estimated wait time. This will issue a REST call with a map of services and attributes to a servlet on the `CustomerController` cluster, which in turn will request an estimate from Work Assignment. It has the following behaviour by default:

- The request uses a single service map. While it may use more than one service map, by default only the first service map in the response is used to check the estimated wait time.
- When the service returns a response, the UI parses the response for a metrics object. If this metrics object is undefined, it is likely because the specific attribute set and priority that will be used in the chat has never been requested. The chat panel remains visible.
- If the metrics object is defined, it is parsed to return the estimated wait time. If this is greater than a customisable value of 600 seconds (10 minutes), or less than 0, the chat panel is hidden; otherwise, it remains visible. If there are no agents logged in on this service, chat is not available; the chatbot is specifically excluded from this consideration.

The request uses the default chatLogon attributes upon page load. To manually send the request from your own code, use the following command: `ewt.requestEwt()`. The chat logon attributes will be converted to a service map instead of an array when the request is sent.

To add more service maps, edit the requestEwt method to include a second service map.

5.12. Removing the Configuration Panel

The configuration panel attached to the home.html page is used to configure the Web UI for demos. It allows you to override the default configuration for a specific browser on a specific machine, and should not be used in production sites. To remove this for a production website:

1. Open the home.html page in a text editor or IDE.
2. Find the element identified as "configLink".
3. Remove or comment out the element.
4. Reload the page. If it worked, the gear icon should no longer be visible.
5. Remove the chatConfigPanel.js and chatConfigPanel.css files from the page.
6. Remove the HTML elements for the panel.

Note: there is currently no way to open the configuration panel from code. It requires an element identified as "configLink".

5.13. Resetting the chat panel upon completion of a chat

When the chat is closed, the default behaviour of the Web UI is to reset the chat panel to login state after a delay of 5 seconds. To change the default behaviour, edit the resetTimer variable inside the webChatConfig.js file to a different value. For example, if you wish to lower this to 3 seconds, change this value to 3000. The actual functionality is defined inside the handleClose method inside webChatSocket.js

5.14. Localisation

The out-of-the-box Web UI uses English. Dates will be presented in the user's locale using the standard [Date.toLocaleString](#) method, e.g. if the Oceana server is on GMT, but the user is on Central European Time (GMT +1), timestamps for chat messages will show in Central European Time. The text direction of the page (e.g. LTR, RTL or blank) will be passed up when opening the chat and shared with the agent client.

5.14.1. Current

In Oceana 3.6.1.0 and later, sections of text that may require customisation may be found in the following files:

File	Affected method/variable	Impact
avayaGlobal.js	browserWarning	This variable lets the user know that their browser does not support required features
chatConfigPanel.js	toggleSecureConnections	This method is called when the configuration panel is used to flip connection security. It is only used in demos or for testing.
	loadCustomData	This is used to load an arbitrary piece of JSON to send with chat messages. This method is only used in demos or for testing.
estimatedWaitTime.js	chatAvailableMsg, chatPossibleMsg, chatNotAvailableMsg, noAgentsAvailableMsg	These are used to let the customer know if chat is available, and if so, how long they will be waiting
webChatLogon.js	gatherDetails	The error message that may be shown if the customer's details are invalid is comprised of several parts that are scattered throughout this method
webChatConfig.js	openingPageText coBrowseSessionFinishedText	These are placeholders and sample messages to show during a chat
webChatUI.js	addEwtToChatTab	This function adds a tooltip to the live chat link to allow the user to see their estimated wait time.
	markEIAsRequired	This function updates a specific label in the login panel to mark it as required
customerJourneyUI.js	subscribe	This is a sample method used to request a customer ID using an email address. If the email address is empty, it displays a text box asking the user to enter an address
customerJourney	parseCustomerIdResponse	This function handles the response for the customer ID.

5.14.2. Legacy (prior to 3.6.1.0)

If you wish to change various system messages displayed in the chat panel, prior to Oceana 3.6.1.0, you will have to edit the following methods:

- webChat.js/chatLogin
- webChat.js/notifyRequestChat
- webChat.js/notifyNewPagePushMessage
- webChat.js/notifyNewCoBrowseSessionKeyMessage
- webChat.js/notifyNewParticipant
- webChat.js/notifyParticipantLeave
- webChat.js/notifyRouteCancel
- webChat.js/quitChat
- webChat.js/onCoBrowseSessionClose
- webChat.js/onCoBrowseStopSuccess
- webChatSocket.js/handleClose
- webChatSocket.js/handleError
- webChatSocket.js/reconnect
- webChatSocket.js/reloadAfterRefresh

You may wish to refactor the messages inside these methods into public variables that may be changed at runtime. If so, put these inside webChatConfig.js, and refer to them in the same manner as the placeholder messages defined inside the file, e.g.

```
// inside webChatConfig.js
loginStartedMessage: "Sending Login Details",

// inside webChat.js/chatLogin
webChat.writeResponse(chatConfig.loginStartedMessage, chatConfig.writeResponseClassSystem);
```

In Oceana 3.6.1.0 and later, these messages are contained inside the webChatConfig.js file.

5.15. Redirecting to a survey post chat completion

After the chat, you may wish to redirect customers to a survey rating their satisfaction with the chat. For the Web UI, this may be done with a redirect to an external survey tool. The best location for a call to this would be inside the handleClose function inside webChatSocket.js.

5.16. Passing FileTransferNotifications through a proxy (pre-Oceana 3.6.1.0)

It is recommended that a reverse proxy is used to hide Oceana from the Web UI. However, FileTransferNotifications currently include a direct download link. There is a mitigation built into the Web UI as a workaround. To edit this, open webChat.js and find the `replaceFileTransferHost` function. The default mechanism is to simply replace the hostname or IP address.

If you wish to replace this entirely, you can do so as follows:

1. Extract the file UUID and workRequestId parameter from the original URL. The UUID will be the last part of the URL before the workRequestId (e.g. /ABCD-1234?workRequestId=87sa6y87dghsuguyef).
2. Create your own customised URL (e.g. https://chatproxy.example.com/filedownload).
3. Append the workRequestId to this URL (e.g. https://chatproxy.example.com/filedownload/ABCD-1234?workRequestId=87sa6y87dghsuguyef).

In Oceana 3.6.1 and later, the Web UI will construct a URL from the file UUID and workRequestId.

5.17. Removing inline scripts and CSS

The out-of-the-box version of the Web UI uses inline scripts (i.e. JavaScript embedded in the page inside a <script> tag) for initialisation, and inline CSS. If your website has a [Content Security Policy](#) that forbids inline JavaScript or styling, these will not work. Fixing this will require refactoring them into standalone JavaScript files. The following HTML pages include inline scripts:

- home.html
- demo.html
- products.html
- services.html
- solutions.html
- videoconferencing.html
- cobrowse.html
- temporarilyAcceptLabCerts.html (this can be safely removed in production)

Refactoring these initialisation scripts is beyond the scope of this guide.

5.18. Setting the authentication token

As part of a feature request for Oceana 3.4.0.1.20 (integration with IBM Watson), an authentication token was added into the Web UI. The purpose of this is to authenticate the customer via a firewall. Generating this token is beyond the scope of this guide.

To set the authentication token, make the following call:

```
chatSocket.authToken = 'abc';
```

The Web UI will then prepend this into every WebSocket message. This token is not persisted by Oceana or forwarded to the agent desktop client, but will be shared with the chatbot adaptor if the Chat is selected for automation.

6. The Customer Journey

The Customer Journey refers to the gathering of context before and during a contact. In the Web UI, this is usually taken to mean gathering attributes as the customer browses your website before deciding to enter the chat.

6.1. Short sequence (prior to Oceana 3.6.1.0)

This is a quick version of how to set up the Customer Journey (prior to Oceana 3.6.1.0)

```
// request a customer ID from CustomerManagement first.
// This may be skipped if you already have one. Alternative versions are below
customerJourneyCommon.requestCustomerId("email", "test@example.com");
// customerJourneyCommon.requestCustomerId("crmId", "1321415");
// customerJourneyCommon.requestCustomerId("phoneNumber", "+353 555 3888123")

// add extra data. These will not be sent unless you update the schema in Oceana Core Data Service.
oceanaCoreData.addDataPair("foo", "bar");
oceanaCoreData.addDataPair("myData", {
  "foobar" : "fizzbuzz",
  "anyOldNumber" : 543,
  "aBoolean" : false,
  "nestedObject" : {
    "array" : [1,2,55]
  }
});

// initialise the context. Must happen AFTER the customer ID has been returned.
// If you are running this on the same page as chat, this must be called *BEFORE* the chat begins
oceanaCoreData.initialise();

// add an attribute. If the context has not been initialised, that will be done in the background.
oceanaCoreData.addAttribute("Language.Gaelic");
```

Alternatively, you may set up the non-routing data as follows:

```
oceanaCoreData.data = {
  "foo" : "bar",
  "myData" : {
    "foobar" : "fizzbuzz",
    "anyOldNumber" : 543,
    "aBoolean" : false,
    "nestedObject" : {
      "array" : [1,2,55]
    }
  }
};
```

6.2. Short sequence (3.6.1.0 and later)

```
customerJourneyUI.setup();

// request a customer ID using the customer's email address
customerJourney.requestCustomerId("email", "test@example.com");

// alternatively, request using social media details. This was available in the CustomerManagement API, but
// there was no method in the Web UI to access it
```

```

customerJourney.requestCustomerIdSocialMedia("facebook", "insertYourSocialHandleHere")

// set a topic. If not set, it will use "Default_Topic"
customerJourney.setTopic("My_Example_Topic");

// a new method to add attributes. The third parameter identifies the service to which to add this attribute,
and is optional. You can repeat this line as often as you like.
customerJourney.addRoutingAttribute("Language", "German", 1);

// add non-routing data
customerJourney.addNonRoutingData("foo", [1,2,3,4]);

// set up the context. If there is already an existing context ID, it will request the previous one.
customerJourney.setup();

// this is the old method for adding attributes, kept for backwards compatibility. The second parameter is
optional; if omitted, the code assumes you wish to insert into the first service.
customerJourney.addAttribute("Service.Sales", 1);

// update the schema in OCDS
customerJourney.updateSchema();

```

6.3. Initialising

To initialise the Customer Journey code, call the following function. This will create a context if the workRequestId/contextId is blank or null; otherwise, it will request the context that has previously gathered.

```
customerJourney.setup();
```

Take note of the following caveats:

- If you are calling this on the same page as chat, make sure that the chat does not open until the request has finished.
- You must have a customer ID generated in Oceana beforehand. See the next subsection for how to do this.
- The out-of-the-box version requires a module to handle UI interactions. This is used exclusively when requesting customer IDs.

6.4. Generating the Customer ID

To generate a customer ID, make a request to the CustomerManagement service with a particular contact detail. The full syntax is listed in the API section at the end of this document. It requires one of the following details:

- Email address
- Phone number
- Social media handle and platform (both values are required). Prior to Oceana 3.6.1.0, there is no out-of-the-box method to handle this in the Web UI.
- CRM ID (the ID of the customer in your CRM system)

This will return their customer ID in Oceana. If the customer does not exist in Oceana, a new entry will be generated for them first.

6.5. Adding Customer Journey attributes

The attributes used by the Customer Journey are gathered through HTML pages that include the customerJourney.js file. These can be added either on page load, or when the user clicks on a particular element in the page. In either case, the new attribute will be added if it does not exist or a maximum value of 10 attributes has not been reached. This maximum value is due to a limitation in the WorkAssignment API; there can only be a maximum of 10 attributes for routing.

6.5.1. Adding an attribute upon page load

Adding an attribute on page load results in that attribute being added for all users that visit the page, making it a useful way to add common attributes. To add attributes on page load, add a method that is called on page load. The following is an example jQuery.ready() function that adds "Product.Oceana" as a routing attribute:

```

<script>
  $(function () {
    'use strict';
    // all three lines are equivalent: they will result in the "Product.Oceana" attribute being
    added to service 1

```

```
        customerJourney.addAttribute('Product.Oceana');
    // customerJourney.addAttribute('Product', 'Oceana');
    // customerJourney.addAttribute('Product', 'Oceana', 1);
    });
</script>
```

To stop adding "Product.Oceana" on page load, remove this script from the page.

6.5.2. Adding an attribute upon clicking an element

Adding an attribute when an attribute is clicked is more targeted than adding it on page load. This can be done by adding an onclick event to an element. The following code snippet is an example of a div element with an onclick event attached:

```
<div onclick="customerJourney.addAttribute('Product.Oceana')">Click here for more details on Avaya Oceana</div>
```

6.5.3. Adding extra data

To add extra, non-routing data in the Customer Journey, call the following function:

```
customerJourney.addNonRoutingData(key, value);
```

e.g. in the following example, add a data pair with key "foo" and value "bar", then a second with key "example" and a JSON object as the value. The key "must" be a String; the value can be any JavaScript object.

```
customerJourney.addNonRoutingData('foo', 'bar');
customerJourney.addNonRoutingData('example', {
  'details' : {
    'first' : 'string',
    'second' : 'string',
    'firstArray' : []
  }
});
```

Note: this function will only update the data locally. If you do not add an attribute after this, the extra data will not be visible in the customer journey.

7. Testing the Web UI

This section details how to test the customer journey through the website to chat:

1. Visit the home page and open the browser console.
2. Click on the "Scopia" link at the top of the page. This will open the videoconferencing.html page. A widget will appear, allowing you to subscribe via email.
3. Enter your email address into the widget and click "Sign me up!"
4. Click on the "View details" button below "Avaya Scopia Desktop and Mobile Applications". This will add "Solution.TeamEngagement" as a routing attribute.
5. Click on the "Products" link at the top of the page. This will open the products.html page, which adds itself as a touchpoint/customer journey element. By default, the touchpoint is the title of the page, i.e. "Products.html". This will not appear in Customer Journey, as there are no attributes added here.
6. Click on the "Solutions" link. This will open the solutions.html page, which adds "Solution.CustomerEngagement" on page load.
7. Click on one of the buttons marked "View details". These do not open anything, but will add an attribute (e.g. the button below "Customer Engagement" will add the attribute "Solution.CustomerEngagement"). Hovering the mouse over the buttons should show the attribute that they will add.
8. Click on the "Services" link. This will open the services.html page, which will add the attribute "Product.Services" on page load.
9. The console should now list the attributes as "Product: [UnifiedCommunication, Services], Solution: [CustomerEngagement].
10. Click the "View details" button at the bottom of the "Professional Services" entry. The console should now include "Service: [ProfessionalServices]" in the attributes.
11. Go to the home page. Click on the "Live Chat" tab on the side. Enter the user's details, and click "Chat Now" to open the chat.
12. The chat panel should now change to the transcript panel. The first message should read "Sending login details".
13. Wait for an agent to join.
14. When an agent joins, the chat controls are enabled. Start exchanging messages with them.

8. Troubleshooting the UI

The Web UI is where symptoms of an internal problem often arise first. This section lists issues that you may face and how to fix them, along with the location of log files that may be useful.

8.1. General troubleshooting procedure for Web Chat

The following steps detail how the chat is opened.

1. Check the browser console. If it reports that the browser couldn't connect to the chat endpoint, this is most likely a certificate issue.
 - a. If it reports a 405 error, it's probably a CORS issue. Make sure that the External Web Domains entry in the OCP Admin is set to "*" or the hostname of your frontend server.
2. Check the CustomerController SVAR logs (*/var/log/Avaya/services/CustomerControllerService*) for EWT issues.
3. Check the CustomerController PU logs (*/var/log/Avaya/dcm/pu/CustomerControllerService*) for anything related to chat.
 - a. If a NumberFormatException occurs, this is due to ORC not returning a number for the Customer ID. In this case, skip to the next step.
4. Check the ORCRestService logs (*/var/log/Avaya/services/ORCRestService*).
5. Check the CustomerManagement service logs on the Context Store nodes (*/var/log/Avaya/services/CustomerManagement*).
6. Check the OCPDataServices logs on the OCP cluster (*/var/log/Avaya/services/OCPDataServices*).
7. Check the OCDS logs on the ContextStore cluster. (*/var/log/Avaya/services/OceanaCoreDataService*).
8. Check the UCM PU logs on the Common Cluster.
9. Check the flow in Engagement Designer.
10. If automated chat is in use, check the AutomationController logs (*/var/log/Avaya/services/AutomationControllerService*).
11. Check the AgentController logs (*/var/log/Avaya/services/AgentControllerService*).
12. Check the UAC logs on the UAC cluster (*/var/log/Avaya/services/UAC*).

To trace a message after a chat has opened:

1. Check the CustomerController PU logs (*/var/log/Avaya/dcm/pu/CustomerControllerService*)
2. If automated chat is in use, check the AutomationController logs (*/var/log/Avaya/services/AutomationControllerService*).
3. Check the AgentController logs (*/var/log/Avaya/services/AgentControllerService*).
4. Check the UAC logs on the UAC cluster (*/var/log/Avaya/services/UAC*).

If the issue turns out to lie in the Web UI, bear in mind the following:

- Browsers are rarely consistent in what JavaScript or CSS they support. Try to reproduce the issue in another browser.
- If the UI shows an alert about certain features not being supported, this is most likely because your browser does not support required features, e.g. Internet Explorer 9 does not support WebSockets. Browsers that do not support WebSockets are not supported.

If the issue does not lie in the Web UI, please include a specific work request ID when providing logs.

8.2. Common errors

The following table details common errors.

Symptoms	Possible Causes	Resolution
Opening a chat results in a connection error, and the browser console reports a "403" Forbidden error.	There is a mismatch between the WebSocket address and the certificate used to authenticate the server.	Check that the Web UI is using the hostname in the certificate and not an IP address. If you are using self-signed certificates, make sure you have accepted them.
Opening a chat results in a connection error, and the browser console reports a "403" Forbidden or "405" Incorrect Method error.	The Web UI server has not been whitelisted in the Omnichannel Admin as an approved origin. By default, no servers are approved. This is separate to the HTTP Security filter on Breeze.	Open the Omnichannel Admin and confirm that the hostname or IP address for the Web UI server is entered in the Omnichannel Admin. For testing purposes, you can allow connections from all origins by entering the wildcard character (*); in a production environment, set this to the hostname or address of your frontend server only.
The chat appears to send a second connection request, which fails. The customer is told that the session key is invalid or expired.	This is caused by a routing issue. If Engagement Designer cannot route the chat to an agent due to e.g. an invalid provider name, the Customer Controller component will close the connection and invalidate the authentication key. However, there is no indication from the Customer Controller that the chat could not be routed, so the Web UI will attempt to reconnect with an expired authentication key.	Check the Engagement Designer logs, and confirm that the provider IDs and names are correct.
When requesting an estimated wait time, the Web UI shows either a HTTP 500 error or the string "EWT Repsonse is empty!" in the console.	This is caused by an error inside the servlet that requests the EWT from Work Assignment. The most likely cause is that it cannot connect to the Work Assignment service.	Check that the WA_URL attribute in the CustomerController properties is set properly, and restart the Omni-Channel cluster.
When requesting an estimated wait time (EWT), the response indicates that chat is not available for a particular set of attributes, even though agents who can handle those attributes are logged in.	The agents that are logged in are not configured to handle chat contacts, or chat agents are not logged in.	Check that there are Web Chat agents available. EWT for the Web UI _requires_ agents whose attributes include the "Channel.Chat" attribute. This is automatically added when making the request.

Chrome tries to reconnect a chat after closing. This only occurs with a multi-node Omnichannel cluster.

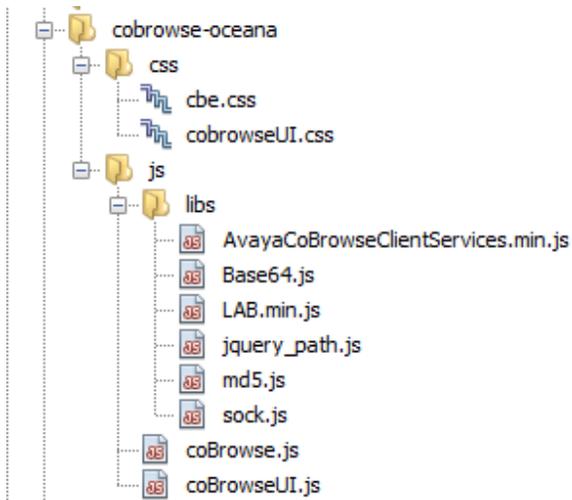
This is a problem with Chrome itself; other browsers close normally. Chrome reports that the connection closes with status 1006 (Abnormal Closure), but Wireshark confirms that it closes with status 1000 (Normal Closure).

This is fixed in 3.5.0.1. If upgrading to 3.5.0.1 is not an option:

1. Update the UI to treat 1006 as a permanent disconnection.
2. When closing normally (agent leave, route cancel, customer close), add the following line:
`chatConfig.dontRetryConnection = true;`

The Co-Browse UI

Included with the sample chat web UI there are also a number of sub-components that provide a reference implementation for implementing Co-Browse in a website. The files associated with Co-Browse are located under `src/main/webapp/cobrowse-oceana`. The files under this path can be seen in the image below:



The `cobrowse-oceana` implementation is designed to be independent from the chat UI, the reason for this being that it also supports Co-Browse sessions with voice contacts. As a result it is stored in its own library directory with its own sub-folders for CSS and JavaScript. All the dependencies located in the `js/libs` are loaded dynamically by `coBrowse.js`, and do not need to be explicitly included on HTML pages (except for `LAB.min.js`). The core code for the Co-Browse included in the sample is included in the `coBrowse.js` and `coBrowseUI.js` code files.

The sample Co-Browse library provided is based on the Observer Pattern. The `coBrowse` object sets up and controls the connection to the Co-Browse snap-in via the Co-Browse SDK (`AvayaCoBrowseClientServices.min.js`). Incoming events from the Co-Browse snap-in are processed in the `coBrowse` object's event handlers, objects interested in these events can subscribe to `coBrowse`. Subscribers will be notified via the event handlers; the `coBrowseUI` object subscribes to these events and updates the user interface based on them.

9. Co-Browse Page Dependencies

The Sample Client expects a number of dependencies to be included on a webpage if it is to be enabled for Co-Browse. It requires implementations of jQuery, jQueryUI (with CSS), and a number of files from the `cobrowse-oceana` folder. Below is a snippet showing the dependencies included on the home.html page to make it Co-Browse enabled:

```
<!-- jQuery and jQueryUI -->
<script src="js/libs/jquery-3.3.1.min.js"></script>
<script src="js/libs/jquery-ui/jquery-ui.min.js"></script>
<link rel="stylesheet" href="js/libs/jquery-ui/jquery-ui.min.css">

<!-- CoBrowse dependencies -->
<script src="cobrowse-oceana/js/libs/LAB.min.js"></script>
<script src="cobrowse-oceana/js/coBrowse.js"></script>
<script src="cobrowse-oceana/js/coBrowseUI.js"></script>
<link href="cobrowse-oceana/css/cbe.css" rel="stylesheet" type="text/css">
<link href="cobrowse-oceana/css/cobrowseUI.css" rel="stylesheet" type="text/css">
```

10. Co-Browse UI Element IDs

The sample Co-Browse UI depends on a number of unique element IDs to be present on a Co-Browseable page. Nearly all of these are added dynamically excluding those related to the `iFrame` and the show Proactive Join Dialog link. It is highly recommend that you check that these IDs remain unique if you intend to consume the sample Co-Browse implementation on a website other than the sample. Their definitions can be found at the top of the `coBrowseUI.js` file:

```
// UI element IDs.
proactiveJoinDialogId : '#proactiveJoinDialog',
proactiveJoinTextId : 'proactiveJoinText',
```

```

proactiveJoinNameInputId : 'proactiveJoinNameInput',
proactiveJoinSessionKeyInputId : 'proactiveJoinSessionKeyInput',

requestControlDialogId : '#requestControlDialog',
requestControlTextId : 'requestControlText',

coBrowseDialogId : '#coBrowseDialog',
coBrowseTextId : 'coBrowseText',

errorDialogId : '#errorDialog',
errorTextId : 'errorText',

// Button IDs used on the dialogs.
joinButtonId : 'join-button',
pauseButtonId : 'pause-button',

```

11. Co-Browse UI Dialogs

The Co-Browse user interface included in the Sample Client is built using the jQueryUI dialog component. A number of dialogs will appear on pages enabled for Voice and/or Chat, these dialogs are added by default to a page that includes the Co-Browse dependencies if the required DIV elements do not already exist; this is handled in the `coBrowseUI.setup` function found in the `coBrowseUI.js` file. By default it will look to create the required DIVs for the dialogs in the top most document on the web page. For example if the `coBrowseUI.js` is loaded into an `iFrame` on a Co-Browseable page it will create the dialogs in the parent document of that `iFrame`. This is done to ensure there is consistent UI behaviour across all of the Co-Browse types. Below is the snippet of code that creates each of the dialogs required for the Sample Client:

```

// Create the Divs needed if they do not already exist.
var element = $(coBrowseUI.proactiveJoinDialogId,
  coBrowseUI.targetDocument).length;
if (element === 0) {
  coBrowseUI.addProactiveJoinDialog();
}
element = $(coBrowseUI.requestControlDialogId,
  coBrowseUI.targetDocument).length;
if (element === 0) {
  coBrowseUI.addRequestControlDialog();
}
element = $(coBrowseUI.coBrowseDialogId, coBrowseUI.targetDocument).length;
if (element === 0) {
  coBrowseUI.addCoBrowseDialog();
}
element = $(coBrowseUI.errorDialogId, coBrowseUI.targetDocument).length;
if (element === 0) {
  coBrowseUI.addErrorDialog();
}

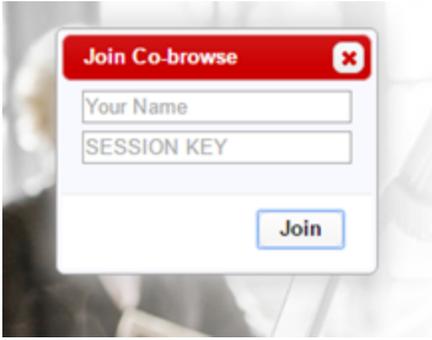
```

In the above code snippet the `coBrowseUI` object checks for the presence of the DOM elements with the required IDs. There is no requirement to define these DIVs, as they will be added dynamically by the `coBrowseUI` object on page load to the target document. It is important to note that if the Co-Browse page is loaded via an `iFrame`, the top most document will need to have the required dependencies for the `coBrowseUI`.

11.1.1. Proactive join dialog

Element ID: **#proactiveJoinDialog**

This dialog is used for Voice Co-Browse scenarios, where there is no way to automatically send a Co-Browse session key to a customer. This dialog will appear on page load and it can be used to proactively join Co-Browse sessions with Voice agents when a session key is supplied over the voice call.



11.1.2. Co-Browse Dialog

Element ID: **#coBrowseDialog**

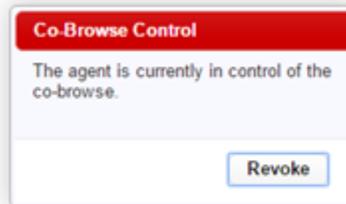
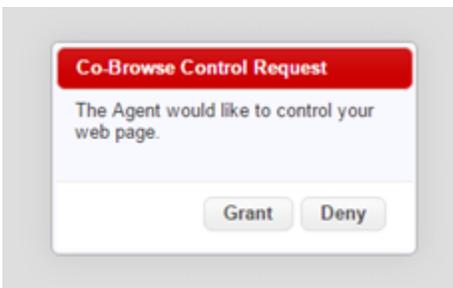
This dialog is the most used dialog and is the first the customer will see when they have joined a session with an agent. It can be used to pause, resume, and stop a Co-Browse session from the customer side. When the session is paused the "Pause" button text will change to the "Resume" action and vice versa. In the top left corner of this dialog the status image (green circle below) will change based on the state, where green indicates sharing is on, and orange meaning sharing has been paused. This dialog will be hidden during control requests and completely closed if the session is stopped for some reason.

11.1.3. Request Control Dialog

Element ID: **#requestControlDialog**

This dialog has two states. It will first appear when a control request from an agent has been received. In this initial state the dialog will have two buttons, one for granting control, and the other for denying it. If the customer chooses to deny control to the agent then the dialog is closed and the Co-Browse dialog will reopen. When the Request Control Dialog appears in this first state it is modal, meaning the customer cannot click anywhere else on the page until they respond to the request.

If the customer grants control to the agent the dialogs interface will update to reflect this choice, the grant and deny buttons will be changed to a revoke control button. When control is revoked from an agent the standard Co-Browse dialog will reappear. If granting, denying, or revoking control fails the customer will be informed via an error dialog.



11.1.4. Error and Info Dialog

Element ID: **#errorDialog**

The error and info dialog is used to provide feedback to the customer in the event of invalid input, reconnects, error scenarios, and other general information such as session closures.



12. Types of Co-Browse Interactions

There are numerous different methods for a customer to interact with Co-Browsing. These are explained in further details below.

12.1.1. Page Push Co-Browse

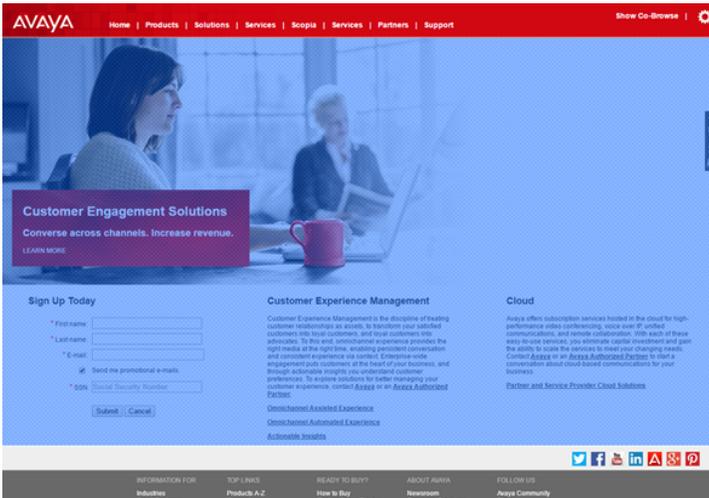
This type of Co-Browse interaction is triggered when an Agent sends a Co-Browse enabled PagePushNotification during a chat. A Co-Browse PagePushNotification can be identified by the presence of the key query path parameter at the end of the URL. Below is an example of a Co-Browse enabled URL that contains a Co-Browse session in the key query parameter:

<http://localhost:8080/CustomFrontendJs/cobrowse.html?key=2065%207368>

When the Sample client receives a PagePushNotification it will check for the presence of the query parameter to determine if the URL is Co-Browse enabled:

```
var sessionKey = avayaGlobal.getParameterByName('key', body.pagePushURL);
```

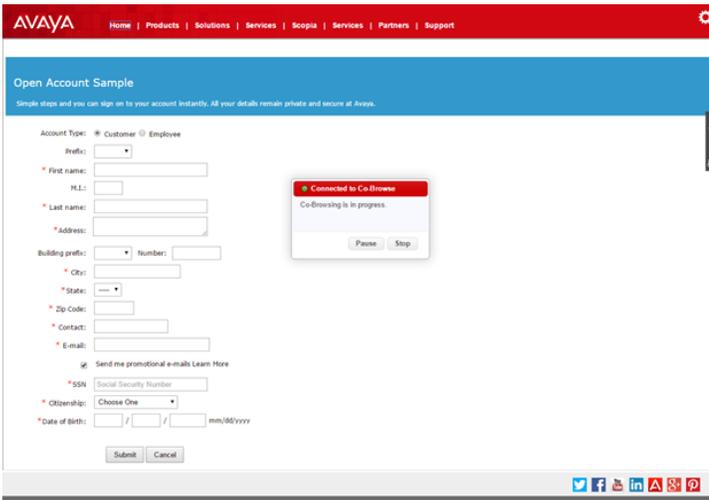
If a PagePushNotification is identified as being Co-Browse enabled the webChat will invoke `coBrowseUI.showCoBrowseIframe(url)` for the instance of `coBrowseUI` loaded on the current page. This will load the URL into an iFrame located in the region highlighted in blue below for the `home.html` page:



When a URL is loaded into the iFrame it will sit above the previous page content until the Co-Browse session is closed by a customer, agent, or when an unrecoverable error condition arises. By default the sample Co-Browse implementation looks for an iFrame defined with the following structure:

```
<!-- COBROWSE -->
<iframe id="cobrowse"></iframe>
```

This iFrame is contained within the DIV element `AvayaCoBrowseWrapper`, the area it is loaded into has fixed dimensions in the Sample Client. Size and behaviour of this iFrame can be configured using standard CSS or more advanced techniques to fit the target website. The Sample Client includes a Co-Browseable (`cobrowse.html`) page that can be used in a Co-Browse enabled PagePushNotification for testing. When Co-Browse enabled PagePushNotification is received for this page it will be loaded into the iFrame similar to the example below:

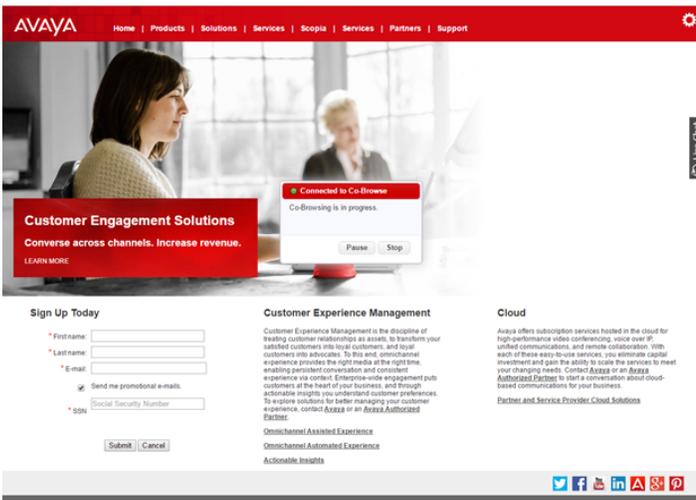


12.1.2. Session Key Push Co-Browse

This type of Co-Browse scenario differs from the Page Push version in that it uses the customer's current page as the target page for the session. The target page must be Co-Browse enabled. A Session Key Push Co-Browse event is initiated when `newCoBrowseSessionKeyMessage` is received during a chat conversation. This message simply contains an open Co-Browse session that the agent has created:

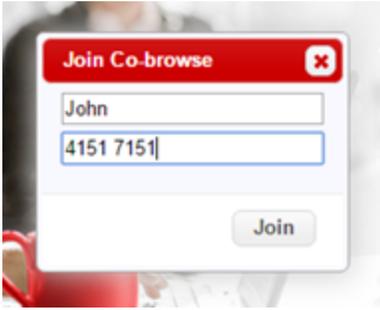
```
{
  "sessionKey" : "5477 5290"
}
```

The `sessionKey` is then processed by the `joinKeyPushCoBrowse` function inside `webChat.js`, which then triggers the underlying call to `coBrowse.joinSession`, passing the received session key to join with. The behaviour of this Co-Browse type is nearly identical to the Page Push version, except that the Co-Browsing is now taking place using the full page rather than just the iframe. It is important to note that trying to switch between Co-Browseable pages in this mode is NOT SUPPORTED in the Sample Client as the chat session is not rebuilt between pages.



12.1.3. Voice Co-Browse

This type of Co-Browse is intended for use with voice customers where there is no means of transmitting a Co-Browse session key automatically. While on a voice call an agent can generate a session key and direct a customer to a Co-Browse enabled page, the key is then verbally exchanged to the customer. Using the Proactive Join Dialog the customer can then enter a name to join under along with the supplied Co-Browse key. When the customer clicks the join button the underlying call to `coBrowseUI.fireJoinRequest` is made. The `webChat` object does not interact in any way with this type of Co-Browse session, instead everything must be customer driven.



There is some basic input processing provided in the `coBrowseUI.fireJoinRequest` function. By default it checks for:

- Does the name field contain a value?
- Does the Co-Browse key field contain a value?
- If the Co-Browse field does contain a value is it in the correct format?

If any of the above evaluate to false then the customer is informed of the field they need to correct. A customer can choose to leave out the space between the Co-Browse key, however then it must be added back as the Co-Browse SDK expects it in the format **XXXX XXXX**. Below is a snippet taken from the start of the `coBrowseUI.fireJoinRequest` function:

```
fireJoinRequest : function() {
  'use strict';

  var name = $('#' + coBrowseUI.proactiveJoinNameInputId, coBrowseUI.targetDocument).val();
  var coBrowseKey = $('#' + coBrowseUI.proactiveJoinSessionKeyInputId, coBrowseUI.targetDocument).val();

  // Basic check of input.
  if (!name) {
    coBrowseUI.createInfoDialog('Invalid Input',
      'Please enter a name to join the Co-Browse session with.',
      coBrowseUI.closeInfoDialog, true);
    return;
  }
}
```

13. Frequently Asked Questions

13.1.1. Where Should I put the Sample Co-Browse Files?

If you plan to consume the Sample Co-Browse implementation directly with no modifications then you should take the `cobrowse-oceana` folder directly and place it in the root of your website. This will make upgrades between releases of the Sample Client simpler.

13.1.2. How do I Upgrade the Co-Browse Sample Files?

If you have consumed the files directly and have not made any modifications then you will simply need to update the files contained under `cobrowse-oceana` folder. You will also need to follow the release notes to understand if there have been changes to other files such as `webChat.js` which references the Co-Browse sample implementation.

In the case that you have made modifications you will need to carefully consider the customisations that you have made and merge them with the newer version using the release notes as a guide.

13.1.3. How do I Configure the Voice Co-Browse Feature?

The Voice Co-Browse `proactiveJoinDialog` no longer appears by default, instead it can be shown using the "Show Voice Co-Browse" link located at the top right of the sample clients `home.html` page:



The code behind the link programmatically creates the `proactiveJoinDialog`. Passing the value `true` will cause it to be shown on the page:

```
$('#showCoBrowseLink').click(function(event) {
    if (!coBrowseUI.checkDialogOpen(coBrowseUI.proactiveJoinDialogId)) {
        coBrowseUI.createProactiveJoinDialog(true);
    }
    event.preventDefault();
});
```

13.1.4. How do I Disable the Key Push Co-Browse Feature

If you only want the Page Push Co-Browse notifications to be processed for a web chat, locate the [handleNotification method](#) inside webChat.js and comment out the line highlighted below:

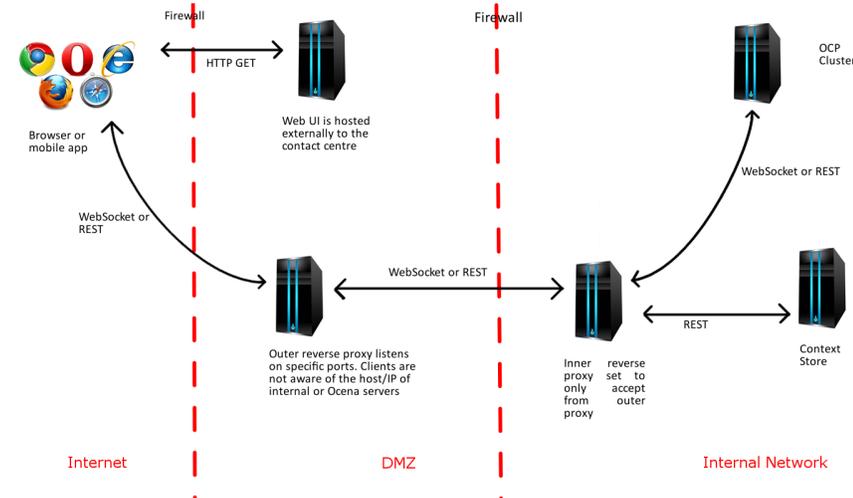
```
} else if (method === chatConfig.jsonMethodRequestNewCoBrowseSessionKeyMessage) {
    webChat.notifyNewCoBrowseSessionKeyMessage(body);
}
```

13.1.5. How do I prevent the user from clicking URLs that will cause them to leave the site during a session?

In Oceana 3.6.0.0, there is a new CSS class defined specifically to act as a selector for this. To use it, apply the "ignoreCobrowse" class to the URL element. You can leave this CSS class blank, or style it as you see fit.

Reference Network Architecture

This section details the reference network topology for Web Chat. It is based around a pair of reverse proxies between the customer client (their browser or the mobile app) and the server. The recommended proxy server is Apache 2.4.25 or later, but any server that supports WebSockets is acceptable. The purpose of this architecture is to hide the Oceana servers from the customer. The following image shows the reference topology



The Web UI is hosted on a separate web server, which does not access the contact centre at all; it mainly acts as a dedicated host for the Web UI. When the customer client requests a chat, it opens a WebSocket directly to the outer proxy.

The outer proxy is configured to hide the address and other details of the inner proxy from the client. It forwards requests on to the inner proxy, which is configured as follows:

- WebSocket requests for the chat are passed to the Omnichannel cluster or node.
- REST requests for the customer's estimated wait time (EWT) are sent to the Omnichannel cluster or node.
- REST requests involving the customer journey are sent to the Context Store cluster or node.
- REST requests to the Co-Browsing service (not shown in the above image) are sent to the cluster or node that hosts the Co-Browsing service.

There is a known issue where FileTransferNotifications from Oceana include direct download links. The Web UI contains a fix for this.

Installing and configuring Apache or any other proxy server is beyond the scope of this guide. Consult the relevant documentation for the proxy server. If you are using Apache, consult the Apache_Proxy_Guide file.

Security Considerations

This section details various security considerations for Web Chat.

14. Ports

The Web UI uses the well-known HTTP and HTTPS ports (80 and 443, respectively). Most firewalls will allow these by default.

The following additional ports are used for the Omnichannel Database and the EmailMangager service. The ports related to the Omnichannel Database are opened in the Windows firewall as part of the installation, and do not need to be manually opened. The ports used for inbound or outbound email may be configured in the Omnichannel Admin. Change these to match the ports on your email server.

Destination Port	Server/Cluster	Purpose	Protocol
1972	Omnichannel Windows Server	JDBC connections to the Omnichannel Database	TCP (can use TLS)
57772	Omnichannel Windows Server	Management Portal for Omnichannel Database	HTTP only
143	Omnichannel cluster in Breeze	Inbound Email (reading from email server)	IMAP over TLS
587	Omnichannel cluster in Breeze	Outbound Email (sending to email server)	POP3 over TLS

15. Secure Transmission

With reference to the section on configuring secure connections for the Web UI, a secure connection to any service must have a valid certificate, and the URL should use a hostname instead of an IP address. The following table indicates the difference, using "FQDN" as a shorthand for the full hostname.

Connection	Unsecure URL	Secure URL
WebSocket for chat	ws://{IP_Address}/services/websocket/chat	wss://{FQDN}/services/websocket/chat
REST call for Estimated Wait Time	http://{IP_Address}/services/CustomerControllerService/gila/ewt/request	https://{FQDN}/services/CustomerControllerService/gila/ewt/request

In general, browsers will refuse to make a WebSocket or REST connection if they cannot guarantee a secure connection, e.g. if the certificate for the server is self-signed, or if the user requests an unsecured connection from a secure page. Secure WebSocket or REST connections are possible from an unsecured page, but the page itself will not be protected from a man-in-the-middle attack; for this reason, serve the Web UI over HTTPS.

It is recommended that requests from the customer be routed via a pair of reverse proxies. For this, Apache 2.4.25 or later is recommended, with the following notes on configuration:

- The outer proxy should be configured to accept from all locations, as the customer client will connect to this. However, it should also be configured to block or hinder denial-of-service attacks at network level. A packet filter may be useful in this role.
- The inner proxy should only accept connections from the outer proxy. This can be done at network level, or at application level.
- Each proxy server must have an individual certificate, with their own private key. This does add overhead when setting up the network, but it adds some redundancy in case a private key is compromised.
- A common cause for errors is that the firewall of the proxy server has not been configured to allow traffic through a specific port.

16. Installing the Automated Chat Server Certificate

This is a guide for installing the TLS certificate for the Avaya Automated Chat server into Oceana, allowing the servers to communicate with Avaya Automated Chat over a secure connection.

16.1. Obtaining the certificate

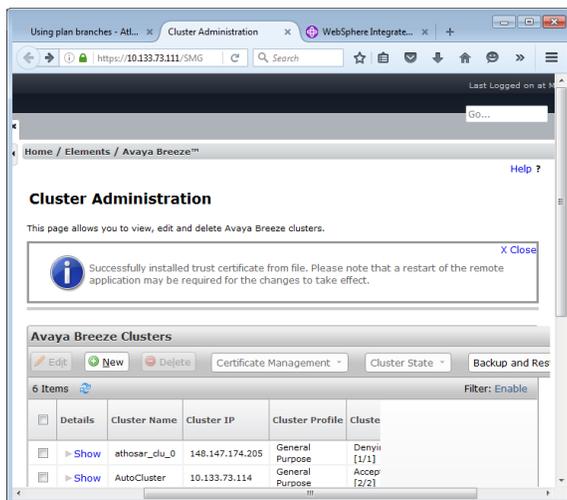
To obtain the certificate for the Avaya Automated Chat server:

1. Access the URL for the server using a browser.
2. Save & download the certificate in .cer format.

16.2. Installation of Automated Chat Server Certificate in System Manager

To install the certificate for Avaya Automated Chat in System Manager:

1. Login to System Manager using valid credentials. Navigate to Home -> Avaya Breeze -> Cluster Administration.
2. Select the Cluster where you would like to install the Automated Chat Server certificate.
3. Click the "Certificate Management" button.
4. Select "Install Trust Certificate" and choose "Certificate to be installed"
5. Click on "Retrieve certificate" and click on the "Commit" button.
6. Ensure that the message in the image below is displayed on the System Manager Cluster administration page.
7. Navigate to Avaya Breeze > Server Administration
8. Select the cluster upon which the certificate is installed.
9. Click the "Shutdown System" button.
10. Choose the "Reboot" option to reboot the cluster.



17. Securing Apache proxy servers

This is a list of tips for configuring Apache to aid securing your network. Some of these will apply to other servers as well. Consult the relevant documentation:

- Keep your servers up-to-date. Apache is under active development.
- Unset the Server header, then set it to a string of your choice. This will return a string stating the type of server, and possibly the version as well. If it is simply unset, the proxy server will fill it in; for Apache, this will be similar to "Apache/2.4.20 (OS) (OpenSSL version)".
- Unset the X-Forwarded-For, X-Forwarded-IP, X-Forwarded-Server and X-Powered-By headers. These are automatically added by the mod_proxy module, which is required for proxies, and contain information about the backend server or the inner proxy, so it is advisable to unset them.
- Unset the Proxy header from any requests. This mitigates the "httproxy" attack, in which a malicious user can redirect outbound traffic to an arbitrary proxy server using a crated HTTP Proxy header in the request. There is already a mitigation for httproxy in Apache 2.4.23 and later, but the Proxy header is undefined by the IETF, and therefore the header serves no purpose. It can be safely disabled.
- Use the Header set Access-Control-Allow-Methods directive to allow the following HTTP methods: POST, GET, OPTIONS, PUT. These are required for various services in Oceana, but the remaining will be disabled by default.
- The following can help in hindering denial-of-service attacks, with the caveat that network-level firewalls (e.g. packet filters) are more effective.
 - Configure the RequestReadTimeout directive to limit the time a client may take to send a request.
 - Configure the ProxyTimeout directive to limit the time a client may use to send requests via a proxy. The default value is 60 seconds.
 - Configure the KeepAliveTimeout directive to limit the time Apache will wait for a subsequent request before closing the connection. The default value is 5 seconds.
 - Configure the LimitRequestBody directive to limit the maximum size of a request. By default, this is 0 bytes, which Apache interprets as being unlimited.
 - Configure the LimitRequestFields to limit the number of fields in a request. The default value is 100 fields.
 - Configure the LimitRequestFieldSize directive to limit the maximum size of a HTTP request header field. It defaults to 8190 bytes.
 - Configure the LimitRequestLine directive to limit the number of bytes that will be allowed on the HTTP request-line. The default value is 8190 bytes.
- Create customised error pages that do not leak any internal stack traces or information about the proxy or backend server. Apache can be set to use them with the ErrorDocument directive.
- Details on configuring strong cipher suites for Apache may be found here: https://raymii.org/s/tutorials/Strong_SSL_Security_On_Apache2.html.

18. Authentication & Session Management

Oceana chat does not include a mechanism for authenticating customers before entering the chat. A customised mechanism must be developed by adopters of the API. This may be done by means of a third-party firewall which examines the "authToken" field inside a WebSocket message. Generating this authentication token is beyond the scope of this guide.

The CORSFilter class on the CustomerController will examine the origin of a chat request; if it does not match the address or FQDN of one or more servers which host the Web UI, the chat will be rejected. This ensures that users cannot open a chat from another site, and can be managed from the Omni-Channel Admin application. This should not be confused with the CORS filter in Oceana, which is managed from the HTTP Security settings on the Avaya Breeze page of System Manager.

A session key has been implemented to hold the state of the customer's session, and to reconnect them if the chat is temporarily disconnected. The chat request will be refused if the session key is null or already in use.

The Web UI uses the browser's sessionStorage to temporarily hold details gathered while logging in or browsing; the sessionStorage in a browser is limited to a single tab, and persists as long as the tab is still open. The following details are stored:

- An email address.
- A telephone number.
- A name.
- A Context Store ID.

19. Authorisation

The customer can only send chat messages into a particular room. If they try to connect or reconnect to an existing chat room, they will be rejected.

Agents may send chat or Page Push Messages to a customer. They may also view the customer's previous interactions, if the customer has previously interacted with Oceana. Page Push Messages may include a Co-Browsing session key that is valid for a particular session.

20. Denial of Service Prevention

There is no single component that will prevent DoS attacks. However, there are several methods that can help in providing mitigation:

- A firewall can be configured to limit the number of simultaneous connections per IP address, with the caveat that this is ineffective against Distributed Denial-of-Service (DDoS) attacks. With reference to the section on the Reference Architecture, the best place to handle this is on the outer proxy.
- The CORS filter on the CustomerController can be configured to only allow chats to be opened from a particular host. If this is set to use the address of the server which hosts the Web UI, then connections that did not originate from that server will be rejected. This helps by preventing attackers from opening chats from unapproved origins. This is configured in the Omnichannel Administration Utility.
- DoS attacks may take the form of sending requests which contain large amounts of data. Consult the section on securing Apache proxy servers for suggestions on hindering this.
- There is a setting in the Omnichannel Admin to limit the number of concurrent chats a customer can make, based upon the entered email address. If the customer reaches this limit, any subsequent chats from that email address will be closed with an Internal Server Error. To configure this limit:
 - Open the Omnichannel Admin.
 - Click on "Web Chat" in the menu in the bottom-left corner.

- Click on "Config" in the menu that appears in the top-left corner.
- Change the value in the "Concurrent Chats Limit per Customer" field.
- As an example, if the setting is limited to 2, a customer with the email address test@test.com may open two concurrent chats, and a third one from this customer will be rejected. However, as the same customer may then open another chat with the address test2@test.com or by leaving the email address blank, this should **not** be relied upon as the sole means of preventing DoS attacks.

21. Data Validation

The Web UI performs some basic input validation, but this is *not* for security purposes. Instead, its purpose is to aid the user experience by e.g. not sending empty messages or letting the customer know that they may have mistyped their phone number.

Data validation is performed using a JSON validation schema on the CustomerController service. This will limit the length of a chat message to 10,000 characters. The server also performs some XSS sanitisation, which strips out potential `<script>` tags from chat messages, and makes sure that email addresses contain valid characters.

Page Push Messages from the agent are validated by the AgentController service. This validates that it starts with `http`, `https` or an empty string, e.g. the following URLs would be considered valid:

- <http://google.com>
- <https://google.com>
- google.com

Access to the Omnichannel database is via a common library, which hinders SQL injection by using parameterised queries.

22. Data Protection

The Omni-Channel Admin for Oceana includes an optional field to link to an external transcript filtering service. This service should be accessed over a secure connection. There is a sample service for testing, but this simply adds the words "been filtered" to the end of each message in the transcript. It must be replaced with a customised one which will filter out sensitive data (e.g. credit card details) before the transcript is persisted to the database.

If the service fails, the transcript is currently persisted in an unfiltered state to the database.

See the Transcript Filtering section for further details on how to use the sample CustomerFilterMessages service to do this.

23. Co-Browse

Enabling secure communications (i.e. HTTPS) on either side of the Co-Browse session will require that both sides run securely. There cannot be a mix of HTTP and HTTPS: if the customer client is running in a secure environment then the agent client must also. The Co-Browse service must also be enabled for secure traffic. To allow for secure connections to a Co-Browse service the `isSecure` flag must be set in `src/main/webapp/cobrowse-oceana/js/coBrowse.js`.

24. Defining and implementing a Content-Security-Policy

Setting the [Content-Security-Policy](#) header may help in hindering XSS (cross-site scripting) attacks. The following header has been tested:

```
Content-Security-Policy "default-src 'self'; script-src 'self' https://code.jquery.com https://maxcdn.bootstrapcdn.com; connect-src *.jquery.com *.bootstrapcdn.com https://1.2.3.4 wss://1.2.3.4; frame-src 'self' https://*.wikipedia.org/"
```

This does the following:

- By default, only resources from the same origin may be used (the 'self' entry).
- Scripts are allowed from the same origin and from the specified CDNs for jQuery and Bootstrap.
- The site may make connections to the following domains/endpoints:
 - `jquery.com`
 - `bootstrapcdn.com`
 - `1.2.3.4` (in this example, the reverse proxy server between the Web UI and Oceana)
 - `wss://1.2.3.4` (a WebSocket connection to the reverse proxy server between the Web UI and Oceana. This is required for chat).
- An iframe may use Wikipedia as it's source (e.g. in the demo.html page), or any page on the same origin as the Web UI.

24.1. Limitations

The out-of-the-box Web UI uses inline event handlers (e.g. the `onclick` attribute on a HTML element), inline scripts for initialisation, and some inline CSS. The above CSP will therefore not work with the out-of-the box Web UI. Refer to the section on removing inline scripts and CSS.

The Android Reference App

25. Disclaimer for Android reference App:

This code is sample code provided as reference implementation and it is up to the end-user to develop production-quality code tailored for their particular business needs.

26. Introduction

The Android Reference App is a sample Android application intended to allow a chat from mobile devices.

27. Capabilities

Text and Emoji Plain text messages and Unicode Emojis The Android Reference App displays any Unicode emoji sent in text messages. Mobile users can use the emoji keyboard on their device to send them.	Image Display static images. The conversation interface supports the display of various image types. Tapping on an image will open it in an image viewer.
GIF GIFs sent via the API will be animated on Android devices. Tapping on a Gif will open it in an image viewer. Note that for animated gifs, the image viewer will display the first frame only. The Android SDK does not include a way for users to choose GIFs to send.	Link Display web links as buttons, Transform links into clear calls to action.
Typing Status Display a typing indicator. Web Messenger can show that the agent is typing with a typing animation. The API allows to specify an agent name.	

28. The archive contains:

- 1) Compiled APK file, ready to run on Android devices.
- 2) The folder with the source code of the project. This is a standard android project. To start working with it, you need to open the "build.gradle" file from the "AndroidChat" folder using the "Open an existing project" menu. Set the target device (emulator) and press "Run app". All dependencies are part of the project. The project is ready to build and run without additional settings.
- 3) PDF file with instructions.

29. QuickStart to your first conversation

29.1. Installing Android Studio

Android Studio is the IDE used to develop Android applications. Skip this section if Android Studio is already installed. To install Android Studio, visit <https://developer.android.com/develop/index.html> and follow the instructions. When this has been completed, import the SDK into Android Studio.

NOTE: This project was developed and tested on version 4.1.12. Recommended to use the same or later.

Prerequisites

To complete the steps below, you must have Android Studio installed & updated, as well as an Android Device to run the sample mobile app (Physical device or Android Virtual Device (AVD)).

Steps

1. Launch Android Studio
2. From the Menu, use File > Open to open the project's / **AndroidChat** subfolder
3. From the Menu, select Build > Clean Project (if the project was built/run and code changes were made)
4. Ensure your Android device is connected (or AVD has been created/configured)
5. From the Menu, select Run > Run App

29.2. Adding the URLs, Work-Flow and Route-point for the app

The Android application allows fully configurable addresses, route-point, work-flow and attributes from demo purposes. To set these values:

1.

- Click the settings icon represented by the three vertical dots, in the top right corner of the home screen.
- Select "Settings", then press "Set Values" to set the required values.
- To add a value for e.g. the webChat address, type the required value into the relevant field.
- Press the "Add" button to add this new value. The field will change colour if this is successful.
- Press the "Clear" button to clear all the previously entered values.
- Press the "Done" button when you are finished.

29.3. Adding Attributes to the app

The Android application allows the manual setting of Attributes for demo purposes. Attributes must be in the format "A"B", e.g. "Service.Sales". To set these values:

- Click the settings icon marked by three vertical dots in the top right corner of the home screen.
- Select "Settings".
- Select "Set Attributes".
- Type the attribute, for example - "Service.Sales" into the provided field, and press the "Add" button. If the entered attribute is formatted correctly then it will be added to displayed list.
- To remove the most recently-added attribute, click the "Remove" button.
- Press the "Done" button when you are finished.

Notes:

- EWT field. The application supports both secure and non-secure connections. The choice depends on the address format — <http://0.0.0.0/> — not a secure connection, https://0.0.0.0 — a secure connection. To fully implement a secure connection, you will need to write the certificate handling code. This test application uses a stub. Details in the section — Customization.
- Customer Controller field. The IP address of the node which allow for the endpoint “/ services / customer / chat”, used to open a web socket — to fill the field the following format should be used — **ws: //0.0.0.0/** or **wss: //0.0.0.0**, depending on the address format, either a secure or not a secure connection.
- Route point — by default, the servers use default, if another route point is configured on your environment, you will need to specify your own.
- If you edited the attribute after saving, then you must press the add button again to save changes.
- Attributes and connection parameters are saved in the application and will be available after restarting the application or restarting the device.
- After completing the filling of the fields, you need to press the DONE button. After that, the application will send a request to the customer service controller to receive the EWT parameter, which will indicate the presence of agents available for chat that correspond to the specified attributes. If the app got the correct response, then a chat icon will appear in the lower right corner. After clicking, a window of additional connection parameters will be displayed. Name, mail, phone. These parameters will need to enter each time you create a chat.

After filling, press the **Submit** button and go to the chat window.

29.4. Chat room

During chat conversation, the client can send text messages and emojis.

File upload is only supported from the agent side. In the case of sending a file, the client receives a link to download the file. By clicking on the link, you will be taken to the default browser on your device to download the file.

To end the chat, just click the back button. You will be taken to the main screen.

If the chat is completed on the agent's side, the client will receive a message about this and the chat window will be automatically closed after 5 seconds. During these 5 seconds, the client can press the back button. In both cases, the client will return to the main screen of the application.

30. Customization and implementation details

30.1. EWT

To check the availability of agents, the customer service controller rest API is used. The application gets the EWT value from the response, if EWT is -1 it means that there are no available agents on the Avaya side. In this case, the chat icon will not be displayed on the screen. A warning will be displayed for the user.

If EWT is greater than -1, then the chat button will be activated on the main screen. However, a value of 0 is undefined and do not guarantee a free agent. This is implemented using an asynchronous call in the inner AsyncPostEWTRequest class. The call chain is as follows -> MainActivity-> onCreate-> requestEWT

Example of getting EWT:

```
String response = null;
if (!addressMap.containsKey("WorkAssignment")) {
    Log.e("Info", "Work Assignment Address was null");
}
```

```

} else {
    String uri = addressMap.get("WorkAssignment") + "/services/CustomerControllerService/gila/ewt/request";
    response = RestConnection.POSTRequest(params[0], uri);
    Log.e("Info", "Work Assignment Address was " + uri);
    Log.e("Info", "Work Assignment response is " + response);
}
return response;

```

30.2. Websocket handling

Websocket handling is implemented in the class - WebSocketService. Conversation between Oceana and chat application is performed based on a standard event – onMessage.

Messages from Oceana side receives in Json format. Parsing of all types of requests performs using one base wrapper class- "CustomerJsonWrapper". Further processing is based on the field – SimpleName

```
className = wrapper.getBody().getClass().getSimpleName();
```

For each valid request exists the model class in the project folder "app\src\main\java\com\lavaya\odonnell3\AndroidChatTemplate_2_3\model\"

Typecast performs before handling, based on model classes. For example:

```
ParticipantLeave pl = (ParticipantLeave) wrapper.getBody();
```

30.3. Changing the appearance of the app

The default appearance of the app may need to be changed to suit your organisation. The following folders contain the files required to configure the appearance:

- app/res/anim " the animation folder. Holds animations.
- app/res/animator " the animator folder.
- app/res/drawable " the images folder.
- app/res/layout " holds XML files that configure the layout of various components.
- app/res/menu " holds an XML file that configures the main menu.
- app/res/mipmap " holds mip-maps (scaled versions of particular images for different screen sizes).
- app/res/values " holds XML files that configure default values for messages.

Below is an example of how to change the background colour of the chat login form:

1. Open the layout folder. This folder contains the XML files that define the layout of various components.
2. Open the content_request_chat_form.xml file.
3. In the Component Tree panel, click on the "Layout View" component. This results in the scroll area for the form being selected.
4. In the Properties panel, click on the box to the right of "background". By default, it points to an image in the images folder. Change this to another image or a colour of your choice.

30.4. Configuring secure connections

Access to servers requiring a secure connection is implemented using a stub. Certificate verification will always successful.

```

if(conn.toString().contains("wss")) {
    SSLContext Cur_SSL_Context = null;
    try
    {
        Cur_SSL_Context = SSLContext.getInstance("TLS");
        Cur_SSL_Context.init(null, new TrustManager[]{new X509TrustManager(){
            public java.security.cert.X509Certificate[] getAcceptedIssuers(){
                return new java.security.cert.X509Certificate[]{};
            }
        }}, new SecureRandom());
        public void checkClientTrusted( X509Certificate[] chain, String authType) throws CertificateException {
        }
        public void checkServerTrusted( X509Certificate[] chain, String authType) throws CertificateException {
        }
    }
} catch (Exception e)
{
    e.printStackTrace();
}
wsClient.setSocketFactory(Cur_SSL_Context.getSocketFactory());
}

```

In order to use a secure connection, you should implement full certificate verification in your version.

30.5. Upgrading from 3.2.1 to 3.2.2.X

To upgrade the Android reference-app from 3.2.1 to 3.2.2.X, navigate to the values/strings.xml file in the project folder and change the path is shown:

Old:<string name="cc_address_tail">/services/websocket/chat</string> New:<string name="cc_address_tail">/services/customer/chat</string>

30.6. Permissions

The app includes the following permissions by default:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

permissions are necessary to function as intended.

31. Testing the Android App

The Android App can be tested using the emulator inside Android Studio, or by deploying to an Android device for testing. Below is a step-by-step guide to test the chat:

1. Open the app.
2. Press the menu icon, marked by three vertical dots, in the top right corner.
3. Set the URLs for the Chat.
4. Set at least two attributes to show the Request Chat icon.
5. Click the Request Chat icon that will appear in the bottom-right of the screen.
6. Select a Chat type.
7. Enter a name and email address.
8. Press "Submit" to start the chat.
9. When an agent joins the chat, type a message into the message box, and press "Send". The message should appear on the right-hand-side of the screen.
10. To close the chat, press the close button.

32. Limitations of the Android App

Multiple concurrent chats are not supported in the sample app. Just one active chat at the same is permitted.

33. Log locations

Logging is done through the logcat library in Android Studio. By default, this prints messages to the development console in Android Studio.

The iOS Reference App

34. Disclaimer for iOS reference App:

This code is sample code provided as reference implementation and it is up to the end-user to develop production-quality code tailored for their particular business needs.

35. Introduction

The iOS Reference App is a sample iOS application intended to allow a chat from mobile devices.

36. Capabilities

Text and Emoji Plain text messages and Unicode Emojis The iOS Reference App displays any Unicode emoji sent in text messages. Mobile users can use the emoji keyboard on their device to send them.	Image Display static images. The conversation interface supports the display of various image types. Tapping on an image will open it in an image viewer.
GIF GIFs sent via the API will be animated on iOS devices. Tapping on a Gif will open it in an image viewer. Note that for animated gifs, the image viewer will display the first frame only. The iOS SDK does not include a way for users to choose GIFs to send.	Link Display web links as buttons, Transform links into clear calls to action.
Typing Status Display a typing indicator. Web Messenger can show that the agent is typing with a typing animation. The API allows to specify an agent name.	

The archive contains:

1) The folder with the source code of the project. This is a standard iOS Swift project. To work with it, you need XCode IDE and any emulator available in XCode.

To get started, open the "Chat Application 1.1.xcodeproj" file from the "iOSChat" folder using the "Open a project or file" menu. Set the target device (emulator) and press "Build and run app"

2) PDF file with instructions.

NOTE: You are able to use real iOS devices to run and debug the application. The real device should be connected to the working MAC by USB cable. Set the real device as target and press "Run" button.

37. QuickStart to your first conversation

Installing XCode

XCode 12 is the IDE required for developing iOS apps. It must be installed on a Mac, and you must have an Apple developer account.

If you do not have XCode already installed:

1. Visit <https://developer.apple.com/xcode/> and click on the "Download" button.
2. Sign into your Apple account, or create one if you do not have one.
3. Follow the instructions on the screen.
4. To launch the project in XCode, open the project folder and double click on the ChatApplication1.workspace file. This will launch the project in XCode.

Prerequisites

To complete the steps below, you must have XCode installed & updated, as well as an Android Device to run the sample mobile app (Physical device or iOS Virtual Device).

Steps

1. Launch XCode
2. From the Menu, use "Open a project or file" > Open the "**Chat Application 1.1.xcodeproj**" file from the "**iOSChat**" folder
3. From the Menu, select **Product > Clean Build Folder** (if the project was built/run and code changes were made)
4. Ensure your iOS device is connected and selected (or VD has been created/configured)

1. Press **Run** button.

Changing the URLs for the App

The URLs and attributes for a Chat must be set once the application has been launched. To set these values:

1. Press the "Settings" button in the top-left corner of the application.
2. Enter URLs and attributes into the specified fields. Each field has a corresponding "Add" and "Delete" button. Once a URL

has been set, it will turn red and new attributes will be added to a list on the bottom of the screen.

1. Press the "<-Chat App" button in the top left corner to set these values and return to the main app screen.

Notes:

1) EWT field. The application supports both secure and non-secure connections. The choice depends on the address format — <http://0.0.0.0/> — not a secure connection, https://0.0.0.0 — a secure connection. To fully implement a secure connection, you will need to write the certificate handling code. This test application uses a stub. Details in the section — Customization.

2) Customer Controller field. The IP address of the node which allow for the endpoint "/ services / customer / chat", used to open a web socket — to fill the field the following format should be used — **ws: //0.0.0.0/** or **wss: //0.0.0.0**, depending on the address format, either a secure or not a secure connection.

3) If you edited the attribute after saving, then you must press the add button again to save changes.

4) Attributes and connection parameters are saved in the application and will be available after restarting the application or restarting the device.

7) After completing the filling of the fields, you need to press the "<-Chat App" button. After that, the application will send a request to the customer service controller to receive the EWT parameter, which will indicate the presence of agents available for chat that correspond to the specified attributes. If the app got the correct response, then a chat icon will appear in the lower right corner. After clicking, a window of additional connection parameters will be displayed. Name, mail, phone. These parameters will need to enter each time you create a chat.

After filling, press the **Ok** button and go to the chat window.

37.1. Chat room

During chat conversation, the client can send text messages and emojis.

File upload is only supported from the agent side. In the case of sending a file, the client receives a link to download the file. By clicking on the link, you will be taken to the default browser on your device to download the file.

To end the chat, just click the back button. You will be taken to the main screen.

If the chat is completed on the agent's side, the client will receive a message about this and the chat window will be automatically closed after 5 seconds. During these 5 seconds, the client can press the back button. In both cases, the client will return to the main screen of the application.

38. Customising the iOS App

Important files and folders

XCode projects have no obvious file structure when viewed outside of XCode. The following are worth mentioning:

1. The ChatApplication1.1 folder contains the majority of the Swift files that make up the application. The files will be sorted into sub-folders when loaded in XCode.

1. Carthage and Pods folders contain the required dependencies for the App.
2. ChatApplication1.1.workspace will launch the workspace for the project in XCode. For more info see XCode Workspaces on

the Apple developer website.

1. ChatApplication1.1.xcodeProj is the application project. The project will be loaded as part of the workspace.
2. Main.storyboard provides a visual representation of the Views and their interconnectedness. Most changes to the visual

appearance or flow of the App will be done here.

1. ChatViewController.swift is the controller for the home screen that is presented when the app is launched.

Changing the appearance of the App

The default appearance of the app may need to be changed to suit your organisation. The layout and appearance of the application can be viewed and modified in the Main.storyboard file.

38.1. Configuring secure connections

Access to servers requiring a secure connection is implemented using a stub in the class - **ChatViewController**. Certificate verification will always be successful. In this case using class - **FoundationSecurity** from framework **Starscream**

```
let pinner = FoundationSecurity(allowSelfSigned: true)
socket = WebSocket.init(request: request, certPinner: pinner)
```

In order to use a secure connection, you should implement full certificate verification in your version.

38.2. EWT

To check the availability of agents, the customer service controller rest API is used. The application gets the EWT value from the response, if EWT is -1 it means that there are no available agents on the Avaya side. In this case, the chat icon will not be displayed on the screen. A warning will be displayed for the user.

If EWT is greater than -1, then the chat button will be activated on the main screen. However, a value of 0 is undefined and does not guarantee a free agent. This is implemented using an asynchronous call in the inner ViewController class. The call chain is as follows -> ViewController-> viewDidLoadAppear-> fillInVariables

Example of getting EWT:

```
if let ewt_addr = addressDictionary?["ewtURL"]{
    let url = ewt_addr+"/services/CustomerControllerService/gila/ewt/request"
    print(url)
    //TODO - build EWT message here
    let map = getEWTFormattedAttributes()
    let json = buildEWTRequest(map: map)
    // print(json)
    sendRestRequest(uri: url, json)
}
```

38.3. Websocket handling

Websocket handling is implemented in the class - **ChatViewController**.

This class implements the interface "**WebSocketDelegate**" and uses it to receive events from WebSocket.

Example of initialization:

```
var request: URLRequest
let hardcodedURL = url! + "/services/customer/chat"
let prepURL = URL.init(string: hardcodedURL)
request = URLRequest.init(url: prepURL!)
let pinner = FoundationSecurity(allowSelfSigned: true)
socket = WebSocket.init(request: request, certPinner: pinner)
socket!.delegate = self
socket!.connect()
```

Conversation between Oceana and chat application is performed based on a standard event – **WebSocketEvent**

Messages from Oceana side receives in Json format. Further processing is based on the field – method:

```
//determine message-type
let received = JSON.init(parseJSON: text)
let body = received["body"]
let method = received["body", "method"]
switch method {   case "newAgentFileTransfer":
    ...
    ...
```

For most of valid request exists the model class in the project folder " \iOSChat\Chat Application 1.1\ "

Typecast performs before handling, based on model classes. For example:

```
func extractAgentMessage(received: JSON) -> ChatMessage{
    let newMessage = ChatMessage()
    newMessage.message = received["body", "message"].string
    newMessage.name = received["body", "displayName"].string
    newMessage.timestamp = received["body", "timestamp"].int32
    newMessage.senderType = "agent"
    return newMessage  }
```

38.4. Upgrading from 3.2.1 to 3.2.2.X

To upgrade the iOS reference-app from 3.2.1 to 3.2.2.X, navigate to the main/ChatViewController.swift file in the project folder and change the path as shown:

```
let fullURL = url!+"/services/websocket/chat"
```

to

```
let fullURL = url!+"/services/customer/chat"
```

38.5. Log locations

The iOS application will log to the console in the XCode IDE. If you want to collect logs from your real device, you need to connect them to MAC by USB cable and run the app using XCode.

39. Testing the iOS App

This section contains a step-by-step guide for testing the iOS App:

1. Open the app.
2. Set the URLs and attributes as described above.
3. Return to the home screen.
4. The application will display a chat icon in the bottom corner after it has requested an Estimated Wait Time. This can take up to

30 seconds.

1. Click on the chat icon. A popup window be displayed, alerting the user that an agent is available.
2. Click "OK". When prompted, enter a username, an email address, a phone number and whether a transcript of the chat is

required.

1. Press "Submit" to start the chat.
2. When an agent joins the chat, type a message into the message box, and press "Send". The message should appear on the

right-hand-side of the screen.

1. To close the chat, press the red "X" button in the bottom-left corner of the chat screen.

40. Limitations of iOS App

Multiple concurrent chats are not supported in the sample app. Just one active chat at the same time.

Transcript Filtering

Chat Transcript filtering is currently performed prior to persisting to the database. The example CustomerFilterMessages artifact provides a sample service that can be deployed outside Oceana (e.g. on a Tomcat server).

For further details on building and deploying a customised version, consult the Transcript Access guide that is bundled with the Web Chat SDK.

41. Enabling transcript filtering

To enable filtering or masking of transcripts do the following:

1. Deploy the CustomerFilterMessages WAR file on a separate Tomcat server.
 - a. You will need to generate a TLS certificate and configure a HTTPS port for Tomcat. This is beyond the scope of this guide.
 - b. Make sure that the firewall on the server that hosts Tomcat allows traffic through the Tomcat ports, e.g. port 8080 or 8443.
2. Open the Omnichannel Administration Utility.
3. Click on "Chat", then "Config".
4. Under "Transcript Filtering Web Service" put the path of the CustomerFilterMessages WAR file. This **should** use HTTPS, i.e. [https://\\${address}/CustomerFilterMessages-\\${version}/rest/filter](https://${address}/CustomerFilterMessages-${version}/rest/filter); however, it will work over HTTP.
5. To configure the different types of filtering, set the "Send Transcript" field to any of the following values:
 - a. 0 (disabled)
 - b. 1 (legacy transcript filtering)
 - c. 2 (enhanced transcript filtering). This option may also result in the transcript being written to a file on the filesystem in addition to being filtered.
6. The certificate for the Tomcat server, or root certificate from the relevant certificate authority, must be imported into System Manager to use HTTPS. To do so:
 - a. Obtain the certificate for Tomcat, or request the root certificate from the relevant certificate authority.
 - b. Open System Manager.
 - c. Click "Inventory", then "ManageElements".
 - d. Select the OCP cluster, then click "Managed Trusted Certificates" under "More Actions".
 - e. Add the certificate.

42. Testing

The following is a sanity testing URL. You can visit this in a REST client, or simply visit in your browser.

```
GET http://${address}:${port}/CustomerFilterMessages-${version}/rest/filter/echo/${message}
Produces text/plain
Simply echoes back the message variable, e.g. "hello".
```

42.1. Filter Request and Response (legacy/basic version)

This requires a REST client, e.g. Postman or the RESTED addon for Firefox. The [/transcript](#) path is added added by the TranscriptFiltering service, so there is no need to put this in the Admin field. In this example, the date fields are null for the sake of convenience; in practice, they would be UTC timestamps.

```
POST http://${address}:${port}/CustomerFilterMessages-${version}/rest/filter/transcript
Content-Type: application/json
Request Body:
[
  {
    "sender" : "Ava",
    "message" : "Welcome to Oceana",
    "date" : null,
    "messageType" : "WELCOMEMESSAGE"
  },
  {
    "sender" : "Dennis Ritchie",
    "message" : "Hello, world!",
```

```

    "date" : null,
    "messageType" : "NORMAL"
  },
  {
    "sender" : "Ava",
    "message" : "I'm going to need some help with this one",
    "date" : null,
    "messageType" : "WHISPER"
  },
  {
    "sender" : "Solid Snake",
    "message" : "This is Snake. Do you read me?",
    "date" : null,
    "messageType" : "COMFORTMESSAGE"
  }
]

```

The POST request will return the following response:

```

Content-Type: application/json
Response Body:
[[{
  "sender": "Ava",
  "message": "Welcome to Oceana been filtered",
  "date": null,
  "messageType": "WELCOMEMESSAGE"
}, {
  "sender": "Dennis Ritchie",
  "message": "Hello, world! been filtered",
  "date": null,
  "messageType": "NORMAL"
}, {
  "sender": "Ava",
  "message": "I'm going to need some help with this one been filtered",
  "date": null,
  "messageType": "WHISPER"
}, {
  "sender": "Solid Snake",
  "message": "This is Snake. Do you read me? been filtered",
  "date": null,
  "messageType": "COMFORTMESSAGE"
}]]

```

42.2. Filter request and response (enhanced)

This is the filter for the enhanced transcript. As with the basic version, it will return the filtered list of messages.

```

[ {
  "sender" : "A: Mr Smith",
  "contactId" : "1",
  "customerId" : "1",
  "workRequestId" : "labdaiuhiu2",
  "message" : "Hello, world!",
  "date" : null,
  "messageType" : "NORMAL",
  "channelType" : "CHAT"
},
{
  "sender" : "C: John Doe",
  "contactId" : "1",
  "customerId" : "1",
  "workRequestId" : "labdaiuhiu2",
  "message" : "This is a test",

```

```
"date" : null,  
"messageType" : "NORMAL",  
"channelType" : "CHAT"  
}]
```

That will return the following JSON response:

```
[{  
  "sender" : "A: Mr Smith",  
  "contactId" : "1",  
  "customerId" : "1",  
  "workRequestId" : "labdaiuhui2",  
  "message" : "Hello, world! been filtered",  
  "date" : null,  
  "messageType" : "NORMAL",  
  "channelType" : "CHAT"  
},  
{  
  "sender" : "C: John Doe",  
  "contactId" : "1",  
  "customerId" : "1",  
  "workRequestId" : "labdaiuhui2",  
  "message" : "This is a test been filtered",  
  "date" : null,  
  "messageType" : "NORMAL",  
  "channelType" : "CHAT"  
}]
```

Web UI API

42.3. Notation

- A field that is marked as **required** is one that must be present; however, this does not necessarily mean that it must have a value.
- If a field must not be blank, then it **must** have a value, e.g. sending up an empty string will result in an error being thrown or a request being rejected.
- Any field listed as x->y is one where y is nested inside x.

43. WebSocket messages

The UI uses a WebSocket for the chat. This section details messages specific to the chat.

Each message contains an authentication token property (referred to as "authToken"). This property can be populated with a token which can be inspected by a firewall application to validate the sender authenticity. This token will also be made available to the Avaya Automated Chat application if the chat is routed through automation. This can facilitate access for the Avaya Automated Chat Agent to data specific to this customer session.

43.1.1. Common fields

These fields are common to all WebSocket messages.

Field	Type	Required	Purpose
apiVersion	String	Yes	Not currently used anywhere. The API requires this to be present, but the value is not validated
authToken	String	Yes	Can be populated with an authentication token which can be used by a 3rd party firewall to validate the message sender. This property is not validated by Oceana. This is prepended to every message inside the sendMsg function inside webChatSocket.js. If your contact centre does not require this, you may leave it blank.
type	String	Yes	Defines the type of message. When sending, this must be 'request'. Messages from the Oceana chat service will either be 'notification' (most messages) or 'acknowledgement' (can be ignored)
body	Object	Yes	The body of the message. For all subsequent messages in this section, only the fields included in the body will be explained.
body -> method	String	Yes	Nested inside the body. Defines how the message will be parsed by the OmniChannel Chat service in the case of a sent message, or how the Web UI parses this in the case of a received message

43.1.2. New Chat Request

This is the very first message sent by the customer after the WebSocket is opened. It is configured and sent by the chatLogin method inside webChat.js, if the customer does not already have an existing session.

Field	Type	Required	Purpose	Persisted to Omnichannel database?
method	String	Yes	Instructs the OmniChannel Chat service to parse this as a new chat request	No
deviceType	String	No	Used by agent client to display an icon, indicating whether the customer is a desktop or mobile client	Yes
routePointIdentifier	String	Yes	Identifies the route point that is used in routing	Yes
workflowType	String	Yes	Defines the workflow for routing chat. Must be either blank, or match the name of a workflow used in routing Chat contacts.	Yes
requestTranscript	Boolean	Yes	Defines whether the user wishes to receive a transcript of their chat via email. The default value is false	Yes
workRequestId	String	No	A context ID that has been generated via a REST call to the Oceana Core Data Service. If not populated here, the OmniChannel Chat service will request one. If populated here, Oceana will associated this context with the chat; the first entry in the context's service map will be updated to include the attributes specified in the intrinsics object	Yes
calledParty	String	No	In chat, represents the page on which the chat originated	Yes
priority	Number	Yes	Used to affect routing. Accepts any number between 1 (highest priority) and 10 (lowest)	Yes
leaseTime	Number	No	Defines the lease time in hours for this contact (i.e. how long it will be held in Context Store). The maximum value is 192 hours; if 0 or null, a default value of 1 hour will be used.	No
intrinsic	Object	Yes	Defines data that is intrinsic to the chat. The fields included in this object are: channelAttribute, attributes, email,	See below

			name, lastName, country, area, phoneNumber, topic and customFields entries	
intrinsic -> channelAttribute	String	Yes	Defines the channel for this contact. Must not be blank; for chat, this is always "Chat"	Yes
intrinsic -> attributes	Array (String)	Yes	Defines the initial routing attributes. These are separate to any attributes gathered via the Oceana Core Data Service before the chat; however, they will be used in routing. May be empty. Invalid attributes (i.e. those that do not match the attributes known to the Omnichannel Database) will be removed	Yes
intrinsic -> textDirection	String	No	Identifies if the page is arranged from left-to-right (LTR), right-to-left (RTL) or unknown(blank). This is used to orient the agent client accordingly.	Yes
intrinsic -> email	String	No	The customer's email address. If empty or the OmniChannel Chat service considers this to be invalid, the customer will not be able to receive a transcript of the chat. The maximum length is 255 characters.	Yes
intrinsic -> name	String	Yes	The customer's first name. Must be present, but can be left blank. The maximum length is 50 characters.	Yes
intrinsic -> lastName	String	No	The customer's last name. To avoid causing conflicts with SMS contacts, this field is not mandatory. The maximum length is 50 characters.	Yes
intrinsic -> country	String	No	Represents the international dialling code for the customer's country, e.g. +353 for Ireland. The maximum length is 10 characters.	Yes
intrinsic -> area	String	No	Represents the area code in the customer's phone number. Bear in mind that not all countries use this field. The maximum length is 10 characters.	Yes
intrinsic -> phoneNumber	String	Yes	Represents the customer's phone number. Must be present due to sharing the API with the Messaging channel, but can be left blank. The maximum length is 32 characters.	Yes
intrinsic -> topic	String	No	Represents a topic for this contact as part of the Customer Management feature. Can also be set in Oceana Core Data Service before the chat. Must not contain spaces or any other character that is invalid inside a URL query parameter. It may contain characters such as ._-~, as long as these are not the first or last characters. For more details, consult the Oceana Core Data Service documentation. If it is not set, Oceana will generate a default one.	No
intrinsic -> customFields	Array (Object)	No	An array of arbitrary Custom Fields (JSON objects with a title and value) that will be stored in the OmniChannel database. Titles must not be blank, must be unique, and have a maximum length of 50 characters. The value of a Custom Field must not exceed 255 characters	Yes
custom Data	Object	No	A structured JSON object which can be used to add any custom properties and will be shared with automation and agent desktop clients	No

Below is the syntax

```
{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "s9d8fcus9",
  "body" : {
    "method" : "requestChat",
    "deviceType" : navigator.userAgent,
    "routePointIdentifier" : "Default",
    "workFlowType" : "workflow_type",
    "requestTranscript" : true,
    "workRequestId" : "342911a23_aas82",
    "calledParty" : "http://www.example.com"
  },
  "intrinsic" : {
    "channelAttribute" : "Chat",
    "textDirection" : "LTR",
    "attributes" : ["Location.Inhouse", "Language.English"],
    "email" : "test@test.net",
    "name" : "Aidan",
    "lastName" : "McCarthy",
    "country" : "+353",
    "area" : "0451",
    "phoneNumber" : "1234567",
    "customFields" : [ {
      "title" : "address",
```

```

        "value" : "Galway"
      }, {
        "title" : "location",
        "value" : "http://www.example.com"
      } ]
    },
    "customData" : {
      "language" : "en",
      "appVersion" : "1.5",
      "device" : "iOS",
      "location" : {
        "latitude" : "12.345",
        "longitude" : "235.34",
        "accuracy" : "1.1",
        "altitude" : "0",
        "altitudeAccuracy" : "1.1"
      },
      "brand" : "Savings"
    }
  }
}

```

43.1.3. Renew Chat Request

This is used to reconnect to the chat. It is configured and sent by the chatLogin method inside webChat.js, if the user is reconnecting. The guid and authenticationKey are assigned by Oceana in the New Chat Notification.

Field	Type	Required	Purpose	Persisted to Omnichannel database?
method	String	Yes	Instructs the OmniChannel Chat service to reconnect the customer.	No
guid	String	Yes	The GUID of the customer's session	No
authenticationKey	String	Yes	The session key	No
requestFullTranscript	Boolean	No	Toggles whether or not the CustomerController will send the entire transcript upon reconnection. The default value is false , i.e. the CustomerController will only send messages that were missed. The expected use case for a value of true is that you are reconnecting after changing pages.	No

The syntax is relatively simple.

```

{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "s9d8fcus9",
  "body" : {
    "method" : "renewChat",
    "guid" : 1,
    "authenticationKey" : "ad89syihuhudwa~efdsepok34_3423",
    "requestFullTranscript" : false
  }
}

```

43.1.4. New Chat Notification

This is used to alert the customer that they have entered the chat, and is processed by the notifyRequestChat method inside webChat.js.

Field	Type	Purpose
method	String	Instructs the OmniChannel Chat service to reconnect the customer.
guid	String	The GUID of the customer's session

authenticationKey	String	The ID of the customer's chat room
wasEmailValid	Boolean	Defines whether or not the OmniChannel Chat service considered the user's email address to be valid. If reconnecting, this will <i>always</i> be set to true to avoid misleading notifications
webOnHoldComfortGroups	Array	An array of messages to play while waiting for an agent. Each group has the following properties: groupName, delay, numberOfMessages, and an array of messages. These are configured in the OmniChannel Administration Utility
webOnHoldComfortGroups -> groupName	String	The name of the comfort group, as defined in the Administration Utility
webOnHoldComfortGroups -> delay	Number	How often they should be played. Measured in seconds
webOnHoldComfortGroups -> numberOfMessages	Number	Defines the number of messages in the array
webOnHoldComfortGroups -> messages	Array	The messages themselves. Each message has a String containing the text, and a Number for the sequence in which to play
webOnHoldURLs	Array	An array of URLs to add to the chat transcript while waiting for an agent. Each group has the following properties: tag, description, holdTime, sequence, and an array of URLs. These are configured in the OmniChannel Administration Utility
webOnHoldURLs -> tag	String	Functions as a name for the group of URLs
webOnHoldURLs -> description	String	Describes the URLs in the Administration Utility
webOnHoldURLs -> holdTime	Number	The interval in seconds between messages
webOnHoldURLs -> sequence	Number	The current order in the sequence. Starts at 1, must be updated after a URL is added to the transcript
webOnHoldURLs -> urls	Array	An array containing the URLs
intrinsic	Object	JSON Object containing any intrinsic values that have changed. Currently only includes the "name" field, which defines the user's display name
intrinsic -> name	String	The user's display name as assigned by the server. Will be used to preface their messages in the transcript panel

New Chat Notification syntax

```
{
  "apiVersion" : "1.0",
  "authToken" : "",
  "type" : "notification",
  "body" : {
    "method" : "requestChat",
    "guid" : 123456,
    "workRequestId" : "34291182",
    "authenticationKey" : "ad89syihuhudwa~efdsepok34_3423",
    "isEmailValid" : true,
    "webOnHoldComfortGroups" : [ {
      "groupName" : "first comfort group",
      "delay" : 45,
      "numberOfMessages" : 2,
      "messages" : [ {
        "message" : "first comfort message",
        "sequence" : 1
      } ]
    } ],
    "webOnHoldURLs" : [ {
      "tag" : "Some tag name",
      "description" : "Test homepage",
      "holdTime" : 30,
      "sequence" : 1,
      "urls" : [ {
        "url" : "http://www.google.ie"
      }, {
        "url" : "http://www.avaya.com"
      } ]
    } ]
  }
}
```

```

    } ]
  } ],
  "intrinsic" : {
    "name" : "Aidan McCarthy",
  }
}
}
}
}

```

43.1.5. New Message Request (text)

This is used to send a message to an agent. It will be defined inside the `sendChatMessage` function inside `webChat.js`.

Field	Type	Required	Purpose	Persisted to Omnichannel database?
method	String	Yes	Instructs the OmniChannel Chat service to parse this as a new message request.	No
message	String	Yes	The actual message text. Must be between 1 and 10000 characters. Note: there is currently a platform limitation where the entire message syntax is limited to 8120 characters. In 3.8.0.0, this has been reduced to 2500 characters to comply with the above platform limit.	Yes
type	String	No	Defines what type of message it is	No
custom Data	Object	No	A structured JSON object which can be used to add any custom properties and will be shared with automation and agent desktop clients. Shared with the New Chat Request	No

New Message Request (text) syntax

```

{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "",
  "body" : {
    "method" : "newMessage",
    "message" : "This is a message",
    "type": "text",
    "customData": {}
  }
}

```

43.1.6. New Message Request (widget response)

This is used to send a response to a widget notification sent by a custom automated agent implementation.

Field	Type	Required	Purpose	Persisted to Omnichannel database?
method	String	Yes	Instructs the OmniChannel Chat service to parse this as a new message request.	No
message	String	Yes	The actual message text that will be displayed to the agent and recorded in the transcript. Must be between 1 and 10000 characters. Note: there is currently a platform limitation where the entire message syntax is limited to 8120 characters. In 3.8.0.0, this has been reduced to 2500 characters to comply with the above platform limit. From 3.8.0.0 limit can be tuned with changing value of <code>IncomingChatMaxDataSize</code> (<code>OutgoingChatMaxDataSize</code>) parameter in <code>cis.SiteParameters</code> .	Yes
type	String	No	Defines what type of message request it is. The possible values are <i>normal</i> or <i>customResponse</i> (used when responding to a widget message from Watson)	No
data	Object	No	Contains fields that are sent to the chatbot adaptor in response to the widget	No
data -> message	String	No	The response to the widget. Must not be blank	No

data -> correlationId	String	No	The automated agent may use this to match this response to a specific widget request	No
customData	Object	No	A structured JSON object which can be used to add any custom properties and will be shared with automation and agent desktop clients. Shared with the New Chat Request	No

New Message Request (widget) syntax

```
{
  "apiVersion": "1.0",
  "type": "request",
  "authToken": "",
  "body": {
    "method": "newMessage",
    "message": "Message goes here?",
    "type": "customResponse",
    "data": {
      "message": "Message goes here?",
      "correlationId": "What type is this?"
    },
    "customData": {}
  }
}
```

43.1.7. Acknowledgement

This is sent from the CustomerController service when the customer's message has been written to the chat room. The main "type" field of this will always be "acknowledgement". The Web UI currently does nothing when it receives this; you may wish to show an icon to let the user know that the message was received.

The **accepted** field shows if the previous message was successfully received. In the current implementation, this is *only* sent when true. The **method** field is required by the API; it is currently only set the "newMessage" to match the New Message Request.

```
{
  "apiVersion" : "1.0",
  "authToken" : "",
  "type" : "acknowledgement",
  "body" : {
    "accepted" : true,
    "method" : "newMessage"
  }
}
```

43.1.8. New Message Notification (text)

This is used to pass a text message to the customer. This will be handled by the notifyNewMessage method inside webChat.js

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a new message response
message	String	The actual message text. Maximum length is 10000 characters. In 3.8, this has been lowered to 2500.
type	String	Defines what type of message it is. In this case, it will be "text".
displayName	String	The name of the agent who sent this reply. This is <i>not</i> the agent's ID or login username
timestamp	Number	UTC timestamp for when the agent sent it
customData	Array (Object)	Customised data populated from the Automated Chat agent

New Message Notification (text) syntax

```
{
  "apiVersion" : "1.0",
```

```

"type" : "request",
"authToken" : "",
"body" : {
  "method" : "newMessage",
  "message" : "This is a reply",
  "displayName" : "Paula Bean",
  "timestamp" : 1238884002,
  "customData": []
}
}

```

43.1.9. New Message Notification (widget)

A widget message may be sent by an automated agent to perform menu-based interaction with the customer. This message contains a data field with details of a menu that the chat client can be customised to display with a GUI element. The format is not defined by Oceana; you must customise the Web UI to match the automated agent.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a new message response
message	String	The actual message text. Maximum length is 10000 characters; in 3.8, this has been lowered to 2500 characters.
type	String	Defines what type of message it is. In this case, it will be "text".
displayName	String	The name of the agent who sent this reply. This is <i>not</i> the agent's ID or login username
timestamp	Number	UTC timestamp for when the agent sent it
data	Object	A structured JSON object which includes details of the widget to implement
data -> type	String	Defines what type of data is being received
data -> correlationId	String	Identifies this specific notification. The client will send this up with the Widget Response type of NewMessageRequest
data -> widgetType	String	Indicates which type of widget is being sent to the client. The current options are "selector" (a dropdown menu) or radio (radio buttons)
data -> text	Array (String)	An array of possible sanitised messages that will be shown to live agents
data -> selectorType	String	Defines the type of selector that will be displayed to the user. Currently supports "button" or "radio"
data -> selectItems	Array (Object)	Possible options for the user
customData	Array (Object)	An array of structured JSON objects which can be used to add any custom properties and will be shared with automation and agent desktop clients. Shared with the New Chat Request

New Message Notification (widget) syntax

```

{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "newMessage",
    "type": "widget",
    "message" : "The agent has pushed a selector",
    "displayName" : "Paula Bean",
    "timestamp" : 1238884002
    "data" : {
      "type": "widget",
      "correlationId": "74272478",
      "widgetType": "selector",
      "text": ["The customer was sent a selector"],
      "selectorType": "bankAccount",
      "selectItems": [
        {

```

```

        "id": "5993dbdc78ec0d4cefbf8060",
        "balance": 110,
        "name": "Personal Account",
        "accountType": "transaction",
        "accountNumber": "123456"
    },
    {
        "id": "5993e5d9eb849c52614d40b3",
        "balance": 356.25,
        "product": "Savings Account",
        "nickName": "My New Car eSaver",
        "accountType": "savings",
        "accountNumber": "123457"
    },
    {
        "id": "5993e615eb849c52614d40b4",
        "balance": 300442.9,
        "product": "Home Loan",
        "accountType": "home loan",
        "accountNumber": "123458"
    },
    {
        "id": "5993e62ceb849c52614d40b5",
        "balance": 1540.2,
        "product": "Platinum Credit Card",
        "accountType": "credit card",
        "accountNumber": "123459"
    }
]
},
"customData" : [
    {
        "type" : "menu",
        "data" : {
            "correlationId" : "74272478",
            "widgetType" : "selector",
            "text" : ["The customer was sent a selector"],
            "selectorType" : "bankAccount",
            "selectItems" : [
                {
                    "id": "5993dbdc78ec0d4cefbf8060",
                    "balance": 110,
                    "name": "Personal Account",
                    "accountType": "transaction",
                    "accountNumber": "123456"
                },
                {
                    "id": "5993e5d9eb849c52614d40b3",
                    "balance": 356.25,
                    "product": "Savings Account",
                    "nickName": "My New Car eSaver",
                    "accountType": "savings",
                    "accountNumber": "123457"
                },
                {
                    "id": "5993e615eb849c52614d40b4",
                    "balance": 300442.9,
                    "product": "MegaCorp Super Affordable Home Loan",
                    "accountType": "home loan",
                    "accountNumber": "123458"
                },
                {
                    "id": "5993e62ceb849c52614d40b5",
                    "balance": 1540.2,
                    "product": "MegaCorp Platinum Credit Card",
                    "accountType": "credit card",
                    "accountNumber": "123459"
                }
            ]
        }
    }
],

```

```

    {
      "type" : "context",
      "data" : {
        "category" : "cars",
        "topic" : "service"
      }
    }
  ]
}

```

43.1.10. Agent File Transfer Notification

This is used to inform the customer that the agent is transferring a file to them. It will be processed by the notifyFileTransfer method inside webChat.js. The download link will be constructed by the getFileDownloadUrl function inside links.js, using the *uuid* and *workRequestId* fields.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a file download link
name	String	The name of the file
address	String	A direct download link to the file that is valid for the customer's current session. If you are using a reverse proxy to hide Oceana from the wider Internet, you will need to update this with the relevant proxy URL. If you wish to generate your own URL, the only parts that must be passed through are the file UUID and the workRequestId Deprecated in 3.6.1, and was removed in 3.7
uuid	String	Identifies the attachment
workRequestId	String	Identifies the contact
mimeType	String	The type of the file
size	String	Size of the file (bytes)

```

{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "newAgentFileTransfer",
    "name" : "black.jpg",
    "address" : "https://w.x.y.z/services/customer/rest/attachment/{fileUUID}?workRequestId={workRequestId}",
    "mimeType" : "image/png",
    "size" : "289"
  }
}

```

43.1.11. New Participant Notification

This alerts the customer about an agent joining the room. This is processed inside the notifyNewParticipant method inside webChat.js.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a newParticipantNotification
agentId	String	The login/LDAP ID of the agent who is joining. If this is an automated agent, this will always be "AvayaAutomatedAgent"
displayName	String	A display label for the agent who is joining. This can be their first name, last name, a combination of the two, or a generic label (e.g. Agent). If this is an automated agent, it will be "Ava"
role	String	Defines the user's role. If they are an active participant , they are driving the conversation with the customer, while if they are a passive participant , they are a secondary agent. The supervisor_observe role is reserved for supervisors that are observing

		/coaching a chat, and supervisor_barge is reserved for when a supervisor barges into the chat.
webOnHoldComfortGroup	Array	A new set of Web-On-Hold messages for this agent's service. These are not persisted to the transcript; their role is to provide filler during a transfer to another service or another agent. The UI currently uses the first group in the array.
numberOfParticipants	Number	Defines the number of users that will be in the chat after this notification
users	Array (Object)	Contains the new list of agents for this chat
user -> id	String	The login/LDAP ID of this agent
user -> name	String	A display name for this agent
user -> type	String	Defines their role. This is used to choose the display icon for this agent. The values here are the same as the "role" entry

```

{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "newParticipant",
    "agentId" : "james",
    "displayName" : "James Bond",
    "webOnHoldComfortGroup" : [{
      "groupName" : "Catchphrases",
      "numberOfMessages" : 2,
      "messages" : ["Bond. James Bond.", "No, Mister Bond. I expect you to die."]
    }],
    "participantRole" : "active_participant",
    "numberOfParticipants" : 1,
    "participants" : [ {
      "id" : "james",
      "name" : "James Bond",
      "type" : "active_participant"
    } ]
  }
}

```

43.1.12. Participant Leave Notification

This alerts the customer about an agent leaving the chat. This is processed inside the notifyParticipantLeave method inside webChat.js. It shares many fields with the New Participant Notification.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a newParticipantNotification
agentId	String	The login/LDAP ID of the agent who is joining. If this is an automated agent, this will always be "AvayaAutomatedAgent"
displayName	String	A display label for the agent who is joining. This can be their first name, last name, a combination of the two, or a generic label (e.g. Agent). If this is an automated agent, it will be "Ava"
role	String	Defines the user's role. If they are an active_participant , they are driving the conversation with the customer, while if they are a passive_participant , they are a secondary agent. The supervisor_observe role is reserved for supervisors that are observing /coaching a chat, and supervisor_barge is reserved for when a supervisor barges into the chat.
leaveReason	String	An uppercase String with a reason for why the agent is leaving the chat. Values are DEFAULT, ESCALATE, TRANSFER and TRANSFER_TO_USER. ESCALATE will be used when an automated agent escalates to a live agent. TRANSFER will be used when transferring the customer to another service.
endChatFlag	Boolean	Toggles whether or not the chat is ending. Will be true if the agent is ending the chat
numberOfParticipants	Number	Defines the number of users that will be in the chat after this notification

rOfParticipants		
users	Array (Object)	Contains the new list of agents for this chat
user -> id	String	The login/LDAP ID of this agent
user -> name	String	A display name for this agent
user -> type	String	Defines their role. This is used to choose the display icon for this agent. The values here are the same as the "role" entry

```
{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "participantLeave",
    "agentId" : "sam",
    "endChatFlag" : false,
    "leaveReason" : "default",
    "numberOfParticipants" : 1,
    "participants" : [ {
      "id" : "james",
      "name" : "James Bond",
      "type" : "active_participant"
    } ]
  }
}
```

43.1.13. Close Conversation Request

This can be sent to close the chat. It has no extra fields.

```
{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "s9d8fcus9",
  "body" : {
    "method" : "closeConversation"
  }
}
```

43.1.14. Close Conversation Notification

This is used to let the customer know that the chat has been closed. The `result` variable is always set to true, and not currently used.

```
{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "closeConversation",
    "result" : true
  }
}
```

43.1.15. Route Cancel Notification

This is used to let the customer know that a chat is being closed due to a timeout in selecting an agent, or an error in the workflow (e.g. an attribute that does not exist is present in the request). The difference between this and the Close Conversation Notification is that the client will explicitly state that it is closing because there are no agents available.

```
{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "routeCancel"
  }
}
```

43.1.16. Page Push Notification

This is used to send a preconfigured URL from the agent to the customer.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as a Page Push Notification
displayName	String	A display label for the agent. This can be their first name, last name, a combination of the two, or a generic label (e.g. Agent)
pagePushURL	String	The URL they are sending. If the URL contains an 8-digit Co-Browsing session key (e.g. 1010 1010), it will be parsed as a Co-Browsing request from the agent.
pagePushDestination	String	Where the URL should be opened. This is a legacy field that is not currently used
timestamp	Number	A UTC timestamp for when the agent sent this
customData	Object	Shared with the NewChatRequest

```
{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "newPushPageMessage",
    "displayName" : "Paula Bean",
    "pagePushURL" : "www.avaya.com",
    "pagePushDestination" : "",
    "timestamp" : 188329439,
    "customData" : [
      {
        "type" : "context",
        "data" : {
          "category" : "cars",
          "topic" : "service"
        }
      }
    ]
  }
}
```

43.1.17. Co-Browse Session Key Message

This is another means by which an agent may initiate a Co-Browsing session. It does not require a URL to be configured in the OmniChannel Administration Utility. The example use case here is that the user would click the "Show Voice Co-Browse" link on the home page, and enter this and their name into the widget that appears. The **sessionKey** field will contain an 8-digit Co-Browsing session key that is used to identify the session

```

{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {
    "method" : "newCoBrowseSessionKeyMessage",
    "sessionKey" : "1010 1010"
  }
}

```

43.1.18. Ping/Pong

JavaScript does not include an API to allow WebSockets to ping the server to confirm that the connection is open, which may result in the customer being unaware of a connection failure until they try sending a message. The Ping message acts as a workaround for this, and by default these are sent every five seconds from the `webChat.sendPing` method. Oceana will send a Pong back for the sake of completion, but the UI does nothing when it receives a Pong. The only difference between the two is that the Pong is of type *notification* and does not contain an *authToken* field. These are fully transient, and will not appear in the transcript at all.

```

{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "",
  "body" : {
    "method" : "ping"
  }
}

```

43.1.19. Is Typing Request

This is used to let the agent know that the customer has started typing. This is a fully transient request; it is not persisted to the transcript at all. The `isTyping` flag is required, and is always set to true; the Chat service will suppress any messages where this is set to false.

```

{
  "apiVersion" : "1.0",
  "type" : "request",
  "authToken" : "s9d8fcus9",
  "body" : {
    "method" : "isTyping",
    "isTyping" : true
  }
}

```

43.1.20. Is Typing Notification

This is sent by an agent to let the user know they are typing. These are transient, and will not appear in the chat transcript.

Field	Type	Purpose
method	String	Instructs the Web UI to parse this as an Is Typing Notification
agentId	String	Identifies the agent who is currently typing
displayName	String	A display label for the agent. This can be their first name, last name, a combination of the two, or a generic label (e.g. Agent)
isTyping	Boolean	Defines whether they are now typing (true) or have stopped (false)

```

{
  "apiVersion" : "1.0",
  "type" : "notification",
  "body" : {

```

```

    "method" : "isTyping",
    "agentId" : "paula",
    "displayName" : "Paula Bean",
    "isTyping" : true
  }
}

```

43.1.21. Error Notification

This is sent by the server to let the customer know that an error has occurred. The error codes are:

- 1: Invalid message (caused by an invalid request, a message longer than 10000 characters, etc).
- 2: Invalid session (session key is invalid or has expired)
- 400: Bad Request (request syntax is invalid)
- 500: Internal Server Error (usually caused by an error between the Customer Controller and another component)
- 503: Not Available (usually caused by the contact centre being put into shutdown mode)

```

{
  "apiVersion" : 1.0,
  "type" : "notification",
  "body" : {
    "method" : "error",
    "code" : 1,
    "errorMessage" : "Invalid message"
  }
}

```

44. REST requests and responses

This section details the syntax for various REST calls and responses.

44.1. Request Estimated Wait Time

This is used to request the estimated wait time for a particular set of attributes. It is called inside the requestEwt method inside estimatedWaitTime.js. The request can accept up to six groups of attributes (marked as "1", "2", and so on), though it is probably simplest to use one only. The WorkAssignment API does not accept more than 10 attributes in a service. For more details, consult the Work Assignment Snapin Developer Guide. You may find that here: <https://www.devconnectprogram.com/fileMedia/download/11a41599-1b9e-4588-8d69-7a5202bdb995>

Service

Field	Type	Purpose	Required
attributes	Object	A JSON object that contains the attributes for this service	Yes
attributes -> attribute	Array	An array of the attribute values. Channel.Chat is always added	Yes
priority	Number	The user's priority. The highest priority is 1; the lowest is 10.	No

```

{
  "serviceMap" : {
    "1" : {
      "attributes" : {
        "Channel" : [ "Chat" ]
      },
      "priority" : 5
    },
    "2" : {
      "attributes" : {
        "Channel" : [ "Chat" ],
        "Language" : [ "English" ]
      }
    }
  }
}

```

```
}
}
```

Estimated Wait Time Response

This is the response for the estimated wait time. By default, only the first group/service ("1") is parsed. The following table lists the fields that may be included in the response for each service. For more details, consult the Work Assignment Snapin Developer Guide. You may find that here: <https://www.devconnectprogram.com/fileMedia/download/11a41599-1b9e-4588-8d69-7a5202bdb995>

Field	Type	Purpose
attributes	Object	A JSON object containing the attributes for this service
attributes -> attribute	Array	An array of the attribute values
priority	Number	The priority that is returned.
metrics	Object	Metrics for this service. If there are no metrics, the reference implementation assumes that chat is available
metrics -> ResourceReadyCount	String	The number of agents who are logged in and could handle this contact
metrics -> ResourceBusyCount	String	Not used in our implementation
metrics -> ProcessingWorkCount	String	Not used in our implementation
metrics -> CompletedWorkCount	String	Not used in our implementation
metrics -> EWT	String	The estimated wait time in seconds
metrics -> WaitingWorkBusyCount	String	Not used in our implementation
metrics -> RollingASA	String	Not used in our implementation
metrics -> ResourceStaffedCount	String	Not used in our implementation
metrics ->ServiceOccupancy	String	Not used in our implementation

```
{
  "serviceMap" : {
    "1" : {
      "attributes" : {
        "Channel" : [ "Chat" ]
      },
      "metrics" : {
        "ResourceReadyCount": "1",
        "ResourceBusyCount": "0",
        "ProcessingWorkCount": "0",
        "CompletedWorkCount": "2",
        "EWT": "0.0",
        "WaitingWorkCount": "0",
        "RollingASA": "2.0494552",
        "OldestWorkWaiting": "0",
        "ResourceStaffedCount": "1",
        "ServiceOccupancy": "0.0"
      },
      "priority" : 5
    },
    "2" : {
      "attributes" : {
        "Channel" : [ "Chat" ],
        "Language" : [ "English" ]
      }
    }
  }
}
```

In the reference implementation, only the **EWT** and **ResourceReadyCount** are taken into account for determining if chat is available. If the EWT is a positive number that is less than 600 seconds, and the ResourceReadyCount is at least 1, then chat is considered to be available.

44.2. Request Customer ID

This is used to request a customer ID from the CustomerManagement service, using specific contact details. In principle, the details can be any of the following, but the default behaviour is to use the customer's email address. If successful, it will return the customer's ID in the Omnichannel Database.

- email=\${emailAddress}
- phone=\${phoneNumber}
- social=\${socialMediaHandle}&platform=\${socialMediaPlatform}
- crmid=\${crmlid}

This request is sent from the requestCustomerId function inside customerJourneyCommon.js. In the reference implementation, this is called when the user clicks the "Subscribe" button after entering their email address on the videoconferencing.html page. The request syntax is as follows:

```
URL: http://a.b.c.d/services/CustomerManagement/rest/customers/customerId?email=${EMAIL}
Method: GET
Response:
{
  "customerId": "${Customer_ID}"
}
```

44.3. Create Context Request

This is the request to create context in Oceana Core Data Service. In the reference implementation, this is called when the customerJourney.setup() function is called and there is no context ID saved to the browser's session storage.

Field	Type	Purpose	Required
data	Object	Non-routing data	No
topic	String	This is used to group interactions within the customer journey, and can be thought of as a "subject" for this contact, e.g. "mortgage". Must not contain spaces or any other character that is invalid inside a URL query parameter. It may contain characters such as .-_, as long as these are not the first or last characters. For more details, consult the Oceana Core Data Service documentation.	No
schema	Object	Defines the schema that will be used for routing	Yes
schema -> Service Map	Object	Contains a list of services that will be used for routing. The syntax here is exactly the same as in the request for Estimated Wait Time	Yes
schema -> Custom erId	String	The customer's ID in Oceana	Yes
schema Strategy	String	Strategy for routing. Acceptable values are "Most Idle" or "FIFO" (First In, First Out). If this request is skipped, Web Chat currently falls back to "Most Idle". For further details, refer to the Work Assignment Developer Guide.	Yes
schema Account Id	String	Added in 3.6.0.0. The purpose of this field is to represent the user's account in your system (as this is more likely to be unique than a phone number). The Web UI has not yet been updated to include this field, but the endpoint API will accept it.	No
groupId	String	Matches the CustomerId	Yes
contextId	String	Identifies the context. If left blank or null, the Oceana Core Data Service will generate a unique ID. If not, the value of this field will be used as the ID	Yes
persistT oEDM	Boolean	Defines whether to make this visible as part of the customer journey. The default value is true	Yes
journey Element	String	Is appended to the URL for auditing purposes. The only accepted entry is "web".	No

```
URL: http://a.b.c.d/services/OceanaCoreDataService/oceana/data/context/schema?journeyElement=${journeyElement}
Method: POST
{
  "data" : {
```

```

    "foo" : "bar"
  },
  "topic" : "",
  "schema" : {
    "ServiceMap": {
      "1" : {
        "attributes" : {
          "Language":["English"]
        },
        "priority":"5"
      }
    },
    "CustomerId":"4802",
    "Strategy" : "Most Idle",
    "AccountId" : "126-BC-76233",
  },
  "groupId":"4802",
  "contextId":null,
  "persistToEDM":true,
}

```

44.4. Context ID Response

This is the response returned by Oceana Core Data Service if the Web UI is requesting a new context. The context ID will be persisted to the browser's sessionStorage for when they enter the chat. If a context ID was supplied during the request, that will be returned here; if not, then the context ID that has been generated by Oceana will be returned.

```

{
  "data" : {
    "contextId" : "wdW0Lc1OQw2ylR1DAwesOw"
  }
}

```

44.5. Customer Journey Context Request/Response

This is the response returned by the Oceana Core Data Service if the Web UI has requested attributes previously associated with a context ID. If attributes have been gathered while waiting for this response, they will be sent to the Oceana Core Data Service as soon as this is returned and parsed.

```

URL: http://a.b.c.d/services/OceanaCoreDataService/oceana/data/context/serviceMap/${contextId}
Method: GET
Response upon success:
{
  "schema" : {
    "ServiceMap" : {
      "1" : {
        "priority" : "5",
        "attributes" : {
          "Language":["English"],
          "Location":["Mobile"]
        }
      }
    },
    "CustomerId":"5530"
  },
  "data":{
    "foo" : "bar"
  }
}

```

44.6. Update Context

This is the syntax for a request to update attributes. The response for this will simply be a HTTP status code, e.g. HTTP 200 for a successful upsert:

Field	Type	Purpose	Required
data	Object	Non-routing data	No
schema	Object	The new schema for routing. Shares the same syntax as that in the Create Context request	Yes
contextId	String	The context ID. Must not be null or blank	Yes
journeyElement	String	Must not be blank. By default, this is set to use the title of the page	Yes

```

URL: http://a.b.c.d/services/OceanaCoreDataService/oceana/data/update/serviceMap/${contextId}?
journeyElement=${journeyElement}
Method: POST
{
  "data" : {
    "foo" : "bar",
    "anExampleBoolean": false,
    "Survey results" : [
      "Excellent",
      "Excellent",
      "Excellent",
      "Okay",
      "Abysmal"
    ]
  },
  "schema" : {
    "ServiceMap" : {
      "1" : {
        "attributes" : {
          "Language":["English"],
          "Location":["Mobile"],
          "Service" : ["Sales"]
        },
        "priority":"5"
      }
    },
    "CustomerId":"4802"
  }
}

```

44.7. Update topic

This is used to update the topic for the customer journey. It has no request or response body.

Parameter	Type	Purpose	Required
contextId	String	Identifies this context in Oceana. Must not be blank	Yes
newTopicValue	String	The new topic value, e.g. "Travel_Insurance". Must not contain spaces or any other character that is invalid inside a URL query parameter. It may contain characters such as ._-~, as long as these are not the first or last characters. For more details, consult the Oceana Core Data Service documentation.	Yes

```

URL: http://a.b.c.d/services/OceanaCoreDataService/oceana/data/update/topic/${contextId}?topic=${newTopicValue}
Method: PUT
No request body
No response body upon success

```