Avaya Breeze® platform Snap-in Development Guide

Release 3.9
Issue 1
January 2024

AVAYA SOFTWARE DEVELOPMENT KIT LICENSE AGREEMENT

REVISED: January 14, 2022

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.

C. 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, NONINFRINGEMENT 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 defend, 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 17 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 AND IMPORT/EXPORT CONTROL.

Licensee shall comply with all applicable laws and regulations, including without limitation those applicable to data privacy, intellectual property, trade secret, and fraud. Licensee is advised that the Technical Information is of U.S. origin and subject to the U.S. Export Administration Regulations ("EAR") and 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 Technical Information 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 Technical Information 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 Technical Information 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.

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.

15.0GOVERNING 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 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.

17.0 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.

18. 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.

Contents

Chapter 1: What you can do with Avaya Breeze® platform

Sample snap-ins

Avaya Breeze® platform supports the following types of snap-ins:

- Call Intercept
- Outbound Calling
- Callable Services
- HTTP-invoked

For more information, see *Avaya Breeze® platform Overview and Specification*.

The Avaya Breeze® platform SDK provides the following sample snap-ins. These snap-ins demonstrate the different concepts and show how to use various capabilities of Avaya Breeze® platform. These snap-ins can also be used as models to test whether your system is correctly installed and configured. The sample snap-ins are not intended for general deployment.

Snap-in Name	Description	Concepts illustrated
HelloWorld	A test snap-in intended for developers to use to verify that Avaya Breeze® platform is working correctly. When either the calling or called party is associated with this snap-in, the message Hello from Avaya Breeze displays on the called endpoint.	Call Intercept Display Manipulation Service Attributes
Whitelist	Routes incoming calls to the dialed user only if the calling number is on a whitelist of numbers designated for the user. If the calling number is not on the whitelist, the call redirects to an alternate number. This snap-in uses Cluster Database as the default database. The snap-in database also supports the use of external databases. The snap-in uses the JDBC Source and JDBC Providers.	Call Intercept Call Redirect Cluster Database Service Attributes
DynamicTeamFormationService	Sends emails or text messages to participants inviting them to join conference calls using Avaya Scopia. Recipients can join the conference by clicking the link in the message.	Scopia conference creation Send SMS Send Email Service Attributes
MultiChanBroadcastService	Sends a broadcast message by email, text message, or voice messages to contacts.	HTTP Servlet Outbound calling Call Event Handling

	This snap-in can be used in Zang-enabled Avaya Breeze® platform environments. For more information about Zang-enabled Avaya Breeze® platform, see *Deploying Zang-enabled Avaya Breeze® platform*.	Play announcement Send SMS Send Email Service Attributes
ClickToCall	Allows users to enter the calling and called numbers on a webpage. The snap-in establishes a call between the calling and the called party numbers. The snap-in displays the call progress on the user interface.	HTTP Servlet Simple web user interface Two-party Make Call Call Event Handling
Callingpolicies	Plays announcements and allows the call to continue to the original number, forks the call to other numbers, redirects the call to another number, or drops the call with an announcement. The caller must use DTMF to select the options. Calling Policies supports Flexible Call Leg Control methods that include serial forking and serial calling operations. This snap-in also demonstrates attribute scoping.	Call Intercept Call Redirect Call Fork (Parallel and Serial) Call Drop Play Announcement Prompt and Collect Service Attributes
CallableService	A test snap-in for Callable Services. The snap-in also demonstrates the use of the message recording function of the SDK.	Callable Service Message Recording Service Attributes
OutboundHttpsSample	Demonstrates correct use of Avaya Breeze® platform facilities to invoke external web services through HTTPS. The snap-in ensures that the latest provisioned trust or identity certificates are always leveraged and ensures that the correct network interfaces on Avaya Breeze sends requests. This snap-in provides a web interface to get the phone number and to display the output of the Zang lookup carrier service.	REST API Invocation Trust / Identity Certificate Use Certificate Change Notification Simple Web UI Zang Carrier Lookup Service Attributes
TaskRepository	An example AuthorizationResource snap-in that, in this context, stores tasks of users. This snap-in has APIs for CRUD operations of tasks that users own. These APIs are protected and only requests with valid access tokens containing appropriate authorization	Authorization Resource REST APIs Service Attributes

	grants can perform relevant operations.	
TaskDashboard	An example AuthorizationClient snap-in that is an OAuth 2.0 client with an intention to provide a user interface for users to view their tasks by retrieving the data from the TaskRepository snap-in. TaskDashboard authenticates users, gets access tokens for them from AuthorizationService and uses these tokens while making task-related queries to TaskRepository.	Authorization Client including user authentication Web User Interface JavaScript invoking REST APIs Service Attributes
AuthorizationSampleBundle	Demonstrates the bundle creation as an SVAR. This sample bundle also shows how snap-ins can be packaged in a bundle and how a snap-in can define dependency on another snap-in.	Snap-in Bundling
CallDeflection	Demonstrates a number of Avaya Breeze® platform features available to snap-in writers. These features include the ability to prompt and collect, send SMS, send email, and add participants to calls. This snap-in can be used in Zang-enabled Avaya Breeze® platform environments. For more information about Zang-enabled Avaya Breeze®, see *Deploying Zang-enabled Avaya Breeze® platform*.	Zang Enabled Breeze Callable Service Add Participant Send SMS Send Email

Chapter 2: Creating a service project

Prerequisite

This chapter assumes that you have the Avaya Breeze® SDK installed in your environment. If you do not yet have the SDK, see *Getting Started with the Avaya Breeze® SDK*.

You must also have JDK 1.8 installed. Only Java version 1.8 is supported.

Project skeletons

The recommended procedure for creating an Avaya Breeze® service project is to use the Maven archetype provided by the SDK. When executed, the archetype creates a skeleton service to act as a starting point for your development.

A service project skeleton can either be created from within Eclipse or by using Maven from the command line. This guide will demonstrate both ways.

Note:

Before you use the Avaya Breeze® 3.9 Maven Archetype, remove the .m2/repositorydirectory and .m2/catalog file from your Maven home directory. You will then have to reinstall the SDK.

Service projects

Project skeleton information

To create a skeleton project, the archetype needs some information from the developer. The group ID, artifact ID, and version are used by Maven to identify the project (aka Maven coordinate). The last two, serviceName and serviceVersion are used by the Avaya Breeze® platform to identify the service.

- Group ID and Artifact ID: Together these uniquely identify your Maven project
- Version: A version string to be used by Maven for your project
- Package: Java package name used to generate the service skeleton code
- serviceName: The name that will identify the service to the Avaya Breeze® platform and System Manager.
- serviceVersion: A version string used in Avaya Breeze® platform for the service in the format *X.X.X.X.X* where *X* is one or more digits. For more information, see the Service Versioning section.

Example Maven values

- Group ID: com.mycompany
- Artifact ID: testService
- Version: 0.0.1-SNAPSHOT
- Package: com.mycompany.testservice

Example Avaya Breeze® platform values

- serviceName: TestService
- serviceVersion: 1.0.0.0.0

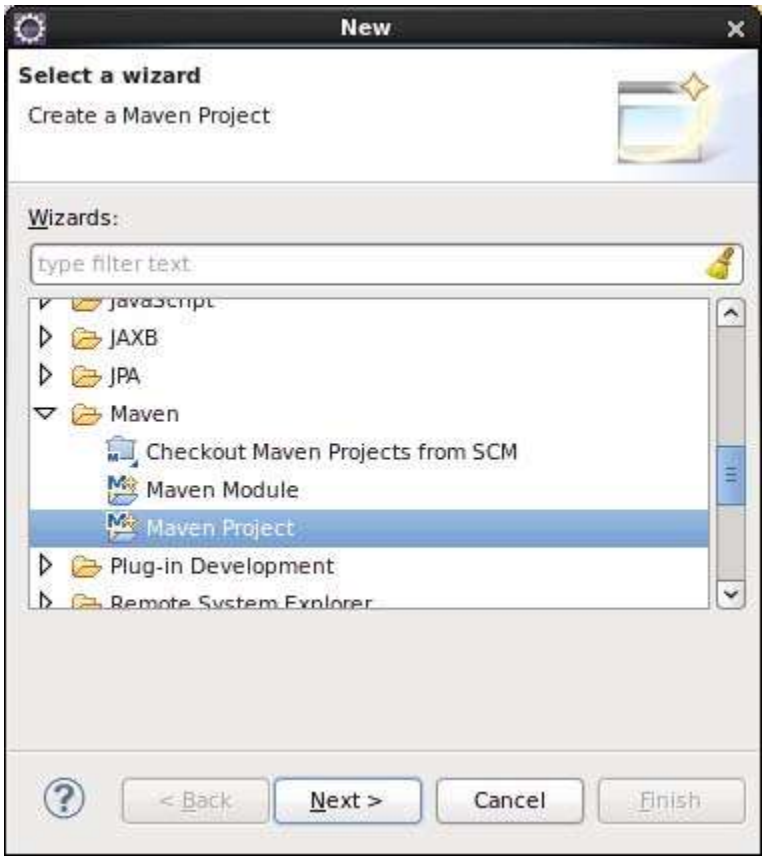
Note:

Service name can contain only the following special characters: hyphen (-) and underscore (_).

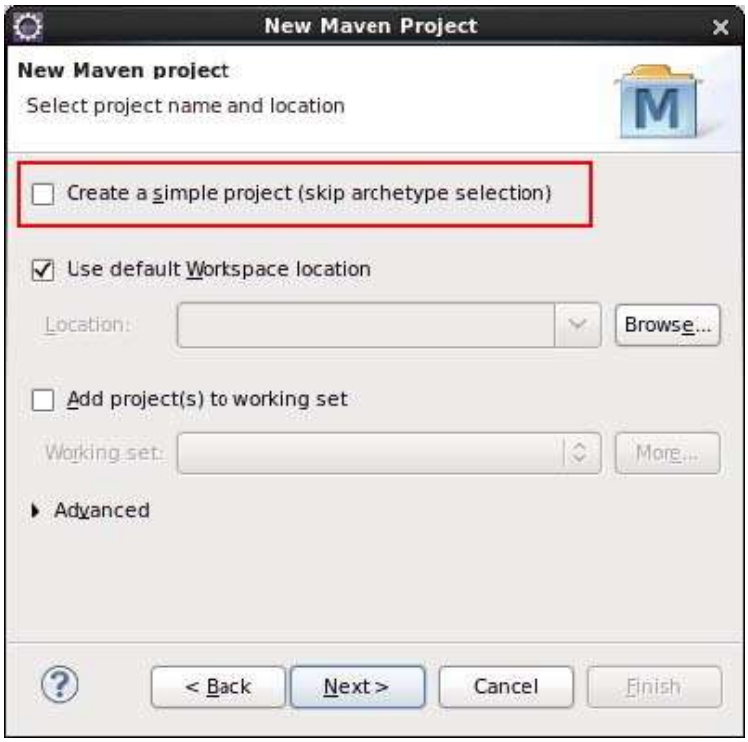
Creating a project skeleton using Eclipse

Procedure

1. Choose **New** > **Other** from the **File** menu.
2. In the dialog that appears, expand the **Maven** group, select **Maven Project**, and select **Next**.

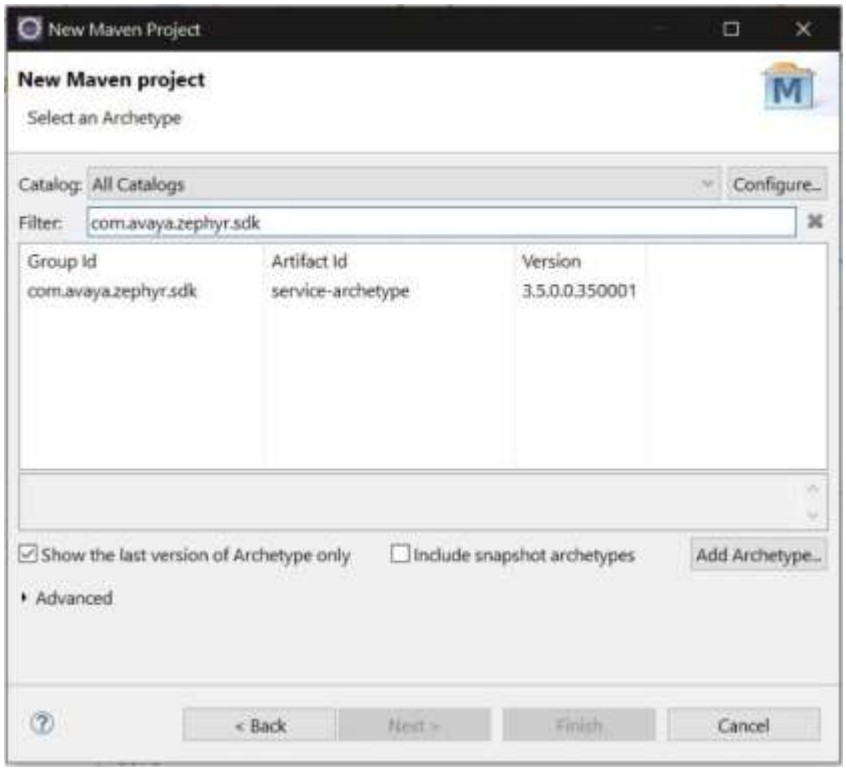


3. Clear the **Create a simple project** check box and click **Next**.

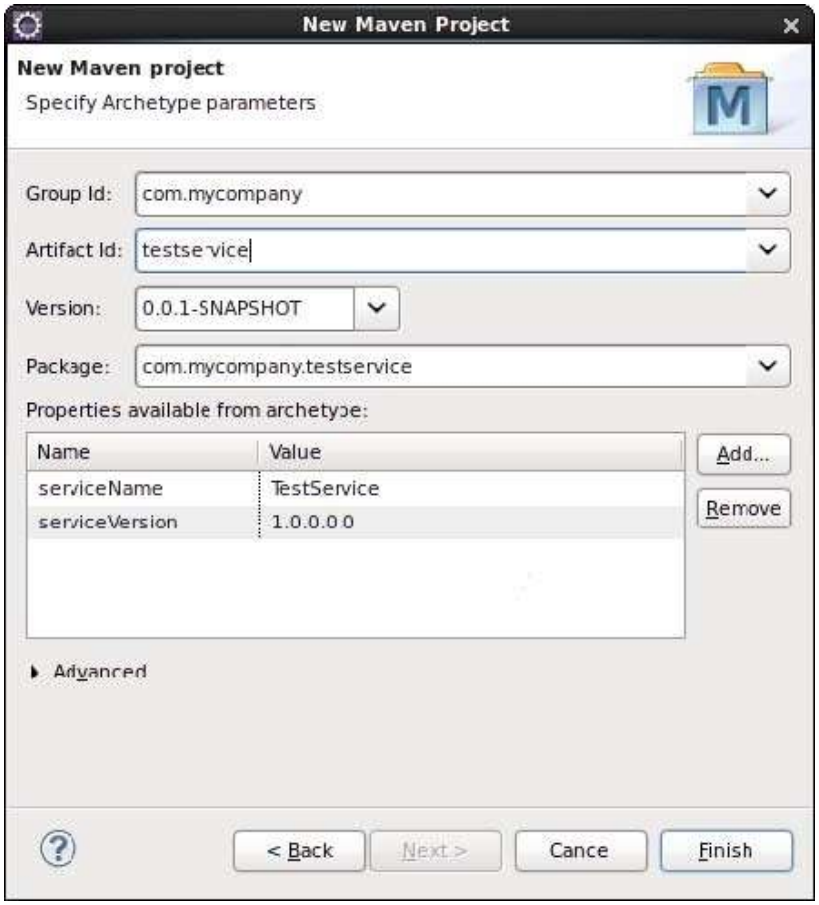


4. Enter com.avaya.zephyr.sdk in the **Filter** field on the archetype selection screen.

5. Select the **service-archetype artifact** and click **Next**.



6. Enter the Maven **Group Id**, **Artifact Id**, and **Version** for the service you will be creating, as well as values for the **serviceName** and **serviceVersion** properties which are used by Avaya Breeze® platform and Avaya Aura® System Manager.



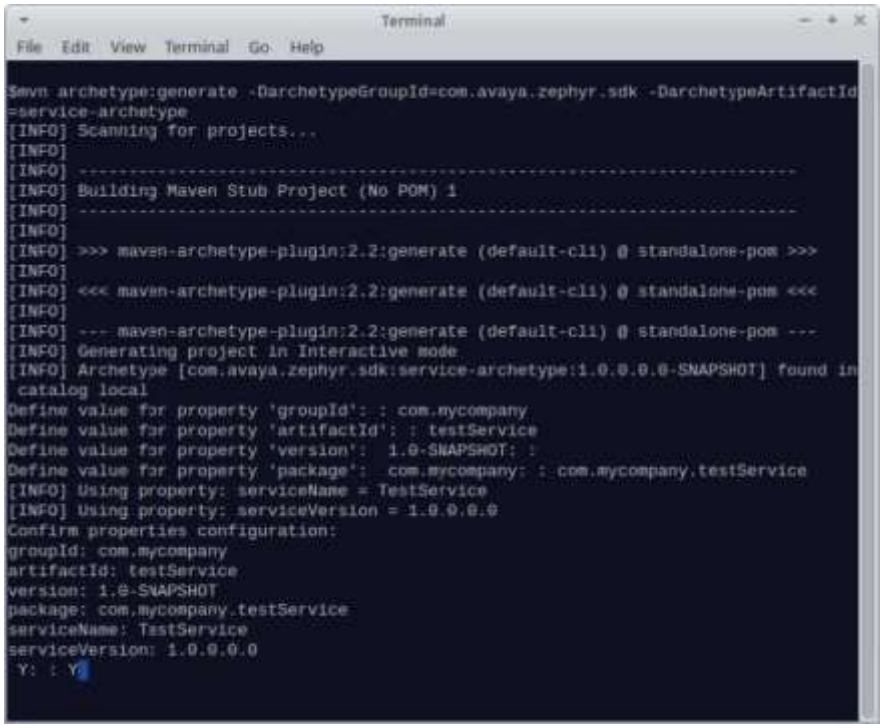
7. Click **Finish**.

Creating a project skeleton using Maven from the command line

Procedure

1. Open a terminal in the directory where you want to create the service project.
2. Run the following command: mvn archetype:generate -DarchetypeGroupId=com.avaya.zephyr.sdk -DarchetypeArtifactId=service-archetype
3. Enter the Maven **group id**, **artifact id,** and **version** for the service you will be creating.

4. If you want to alter the **serviceName** and **serviceVersion** properties from their defaults, enter N at the prompt that appears and reenter the **group id**, **artifact id**, **version**, along with the **serviceName** and **serviceVersion**.
5. Enter Y to confirm that your selections are correct.



Generated project structure

After running the archetype, you will have a parent project which contains 3 modules:

- SVAR module: creates a SVAR file which is the component installed on System Manager and Avaya Breeze® server. It contains the application EAR as well as additional Avaya Breeze® platform specific configuration.
- EAR module: creates an EAR file containing the WAR. Service developers should not need to modify anything in this package.
- WAR module: contains the java source code for the service as well as container configuration.

Note:

Eclipse may display warnings about missing XML schemas in the generated project. These do not affect the service build and can be ignored. XML schema warnings can be disabled in Eclipse by opening **Window** > **Preferences**, navigating to **XML** > **XML Files** > **Validation**, and selecting **Ignore** for **No grammar specified** and **Missing root element**.

Project request types

The archetype sets the service up to handle three types of requests.

Request Type	Details
Calls	Call processing
HTTP JAX-RS	REST web service
HTTP Servlet	HTTP web service or web page

A service does not typically handle all three types of requests. If it does, no trimming is necessary. If not, the one or two unused request types should be trimmed from the project.

Note:

The service archetype does not have any impact on a snap-in's ability to handle Eventing Framework or Collaboration Bus messages.

Trimming project request type: Calls

About this task

If your service does not handle calls, perform the following steps to trim it from the project. Find these files under the WAR module created by the archetype.

Procedure

1. Delete MyCallListener.java (under src/main/java/<yourpackagename>)

   Your package name is determined by the group ID and the artifact ID that was entered when setting up the archetype. Using the example values your package name would be com.mycompany.testservice.
2. Edit the CARRule.xml under src/main/resources and remove the“SequencedServiceRule” and "TerminatingServiceRule" related tags.
3. Delete sip.xml from src/main/webapp/WEB-INF directory.
4. Edit the web.xml under src/main/webapp/WEB-INF and remove the "“servlet” , “servletmapping” and "listener" tags.

Trimming project request type: HTTP JAX-RS

About this task

If your service does not handle HTTP REST Requests, trim it from the project. Find these files under the WAR module created by the archetype, under src/main/<*yourpackagename*>.

Procedure

1. Delete MyApplication.java
2. Delete MyResource.java

Trimming project request type: HTTP Servlet

About this task

If your service does not handle HTTP Servlet Requests, trim it from the project.

Procedure

Delete MyServlet.java

Find this file under the WAR module created by the archetype, under src/main/<*yourpackagename*>.

Platform listener

The platform listener is a Listener interface through which a snap-in is informed of changes in the status of the platform components. In order to receive these events, a snap-in must implement this listener and the snap-in must also annotate the class with @ThePlatformListener as shown below:

```
@ThePlatformListener public class SamplePlatformListener extends PlatformListenerAbstract
    {
        @Override
        public void clusterDbStateChanged(final ClusterDbStateChangeEvent
clusterDbStateChangeEvent)

        {

        }

        @Override
        public void certificateStoreUpdated()
        {
        }
    }
```

For more information, see the PlatformListener and SamplePlatformListener classes in the Avaya Breeze® platform SDK Javadocs.

Call listener implementation

The heart of a call intercept or outbound calling service is the Java code that handles calls passing through the service. Open the MyCallListener.java file. Using the example values above,

it would be located at testService-war/src/main/java/com/mycompany/testservice/MyCallListener.java. Because this class is annotated with @TheCallListener and extends the CallListenerAbtract class, the framework will invoke the callIntercepted method each time a new call enters the service.

Example

This is an example of a very simple call listener that logs calls entering it and allows them to continue as dialed.

```
package com.mycompany.testservice;

import com.avaya.collaboration.call.Call

 import com.avaya.collaboration.call.CallListenerAbstract;

import com.avaya.collaboration.call.TheCallListener;

import com.avaya.collaboration.util.logger.Logger;

@TheCallListener

public class MyCallListener extends CallListenerAbstract {

      private static Logger logger = Logger.getLogger(MyCallListener.class);

                public MyCallListener()

                  {

                  }

                  @Override
                  public final void callIntercepted(final Call call)
                  {
                        logger.info ("Call from " + call.getCallingParty().getHandle() +
                              " entered the service!");
                              call.allow();
                  }
}
```

If your snap-in is a call intercept snap-in, your callIntercepted implementation should call one of the “call verb” methods, allow, divertTo, drop, or suspend. For parallel forking use cases, you can optionally call "addParticipant" in addition to “divertTo” or “allow”. Additionally, for the sake of consistent operation, the call verb method should be called last in the callIntercepted method. If a “call verb” is not called explicitly, the allow method will be called implicitly.

If your snap-in is a callable snap-in, the callIntercepted implementation holds true except that:

- The “allow” verb is not supported for callable snap-ins. This is because the snap-in itself is the ultimate destination of the call, so it does not make sense to “allow” the call to proceed to its original destination.
- Calling “addParticipant” is allowed without additionally invoking “allow” or “divertTo”. This will have the same effect as invoking “divertTo”.

More information on using the Avaya Breeze® platform API to handle calls is presented later in the document. For now, however, let us look at building and packaging the service.

Life Cycle Management

The ServiceLifeCycle interface of the Avaya Breeze® platform SDK defines service initialization and destroys callback methods for the purpose of startup and cleanup, respectively. A snap-in must define a class that implements the “init” and “destroy” methods of this interface, and it must be annotated with “ServiceLifeCycle”. See the “ServiceLifeCycle” Interface in the Javadoc for complete information.

It is important to note that the service life cycle relies on the well-defined initialization and cleanup capabilities of the CallListener interface and is most suited for use with a snap-in that implements call processing functionality. Therefore, a snap-in that makes use of the service life cycle will need to follow the procedures for implementing the CallListener, as described elsewhere in this guide. These procedures include implementing either the CallListener interface or extending the CallListenerAbstract class; they also include providing the sip.xml and CARRule.xml file as part of your project. The Maven archetype included with the SDK provides the framework for the various call processing pieces mentioned here.

In addition to the service life cycle capabilities provided by this interface, there are a variety of standard J2EE life cycle capabilities available to the snap-in developer. One of these other approaches may be more appropriate for a snap-in to use when no call processing functionality is needed. The suitability of any particular approach will depend on the specifics of the snap-in being developed. Several examples are provided below to demonstrate other life cycle options. Additional information about these J2EE elements is widely available on the Internet.

HTTP Servlet

A snap-in implementing an HTTP Servlet can implement init/destroy methods similar to the following:

```
@WebServlet(value = "/MyServlet", loadOnStartup = 1)
public class MyServlet extends HttpServlet
(
      private Logger logger = Logger.getLogger(MyServlet.class);
      public void init() throws ServletException
      {
            logger.info("MyServlet init";
      }
      public void destroy()
      {
            logger.info("MyServlet destroy");
      }
}
```

JAX-RS Resource

A snap-in implementing a JAX-RS resource can do something like this:

```
@Startup
@Singleton
@Path("/myResource)
public class MyResource
{
	private Logger logger = Logger.getLogger(MyResource.class);

	@PostConstruct

	public void startup()

	{

	logger.info("Resource startup");

		@PreDestroy
		public void shutdown()
		{
			logger.info("MyResource shutdown");

		}
	}

}
```

The "MyResource" class shown is generated by default by the Avaya Breeze Maven Archetype. If nothing is modified, this class can be invoked at a URL such as: https://{myBreezeServerAddress}/services/myService/ws/myResource.

Everything up to myService is not specified by the snap-in; it is the default path used by the Avaya Breeze platform to route incoming HTTP request to snap-ins. The last part "myResource" is declared in the "@Path" annotation on the MyResource class. The "ws" portion of the URL comes from another generated class called "MyApplication". This class has an "@ApplicationPath" annotation with the value "ws". All of these class names and paths can be modified. There is only one ApplicationPath annotation allowed for a given snap-in. There can be multiple resources defined with unique Path annotations underneath the ApplicationPath.

Enterprise Java Bean

EJBs can use the following life cycle construct:

```
@Startup
@Singleton
public class MyBean
{
	private Logger logger = Logger.getLogger(MyBean.class);
	@PostConstruct
	public void startup()
```

```
    {
      logger.info("MyBean startup");
      }
      @PreDestroy
      public void shutdown()
      {
            logger.info("MyBean shutdown");
      }
}
```

Building and packaging the service using Eclipse

Before you begin

Note:
An internet connection is required the first time a service is built after installing the SDK as some libraries need to be downloaded from the Maven central repository.

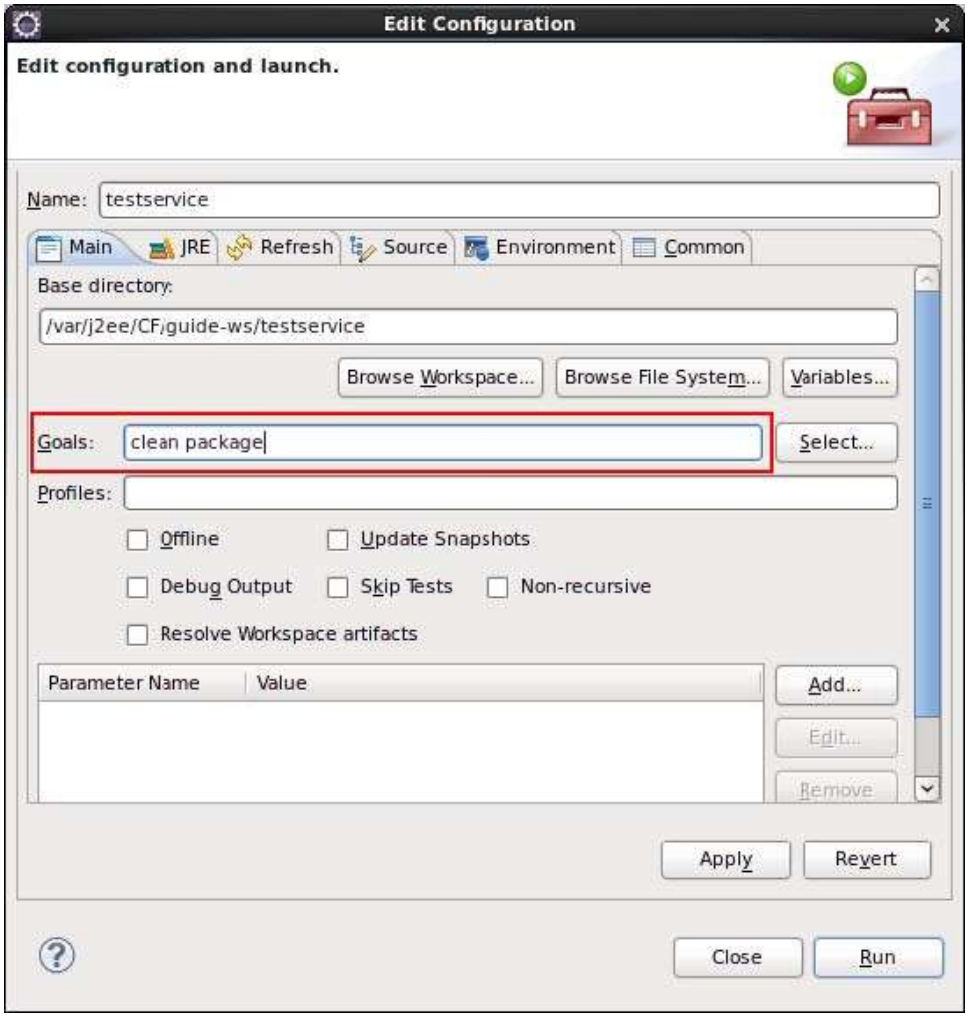
Procedure

1. Right click on the parent project (testService in the example above) in the project explorer and click **Run as** > **Maven build...**.
2. In the window that appears, enter clean package in the **Goals** field and select **Run**.

If the build fails, fix any errors, and rebuild. If the build succeeds, a SVAR archive will be placed in the SVAR module's target directory.

Note:

You may need to refresh the project for the target directory to appear in eclipse.



Building and packaging the service using Maven from the command line

Before you begin

Note:
An internet connection is required the first time a service is built after installing the SDK as some libraries need to be downloaded from the Maven central repository.

Procedure

From the command line, run mvn clean package from the root directory of the parent project.

If the build fails, fix any errors in the source code, and rebuild.

If the build succeeds, a SVAR archive will be placed in the SVAR module's target directory (testservice-svar/target if using the values from the project creation example).

Note:

You may need to refresh the project for the target directory to appear in eclipse.

Avaya Breeze® platform and third-party Jars

Many third-party Jars are used by the platform and are available to use by snap-in developers. This section lists Avaya Breeze® platform and third-party jars along with the version. Version for Avaya Breeze® platform jars are always the same as that of Avaya Breeze® platform. To ensure compatibility, these are the only versions of jars that should be used. Execute “swversion” to retreive the value for <platform-version> and <management-version>.

Jars in lib/ext

- accessors-smart-1.2.jar
- activation-1.1.jar
- aopalliance-1.0.jar
- apache-cassandra-2.1.20.jar
- archive-jaxb-<platform-version>.jar
- asm-5.0.4.jar
- asm-common.jar
- asmmgmt_trust_cli-<management-verion>.jar
- AsmEventCodes.jar
- avaya-commons-io-3.0.jar
- avaya-commons-lang-3.0.jar
- avaya-logging-client-0.0.4.jar
- avaya_logging_formatter.jar
- avaya-periodic-retention-logging.jar
- bc-fips-1.0.2.3.jar
- bcpkix-fips-1.0.5.jar
- bctls-fips-1.0.12.2.jar
- car-xsd-<platform-version>.jar
- cassandra-driver-core-3.8.0.jar
- checker-qual-3.37.0.jar
- commons-codec-1.8.jar
- commons-collections-3.2.1.jar
- commons-configuration-1.9.jar
- commons-dbcp-1.4.jar
- commons-exec-1.1.jar
- commons-io-2.1.jar
- commons-lang-2.6.jar
- commons-lang3-2.1.jar
- commons-pool-1.5.4.jar
- concurrentlinkedhashmap-lru-1.3.2.jar
- dom4j-1.6.1.jar
- error_prone_annotations-2.21.1.jar
- failureaccess-1.0.1.jar
- gson-2.2.2.jar
- guava-19.0.jar

- guava-32.1.3-jre.jar
- guice-4.0.jar
- guice-throwingproviders-4.0.jar
- hamcrest-all-1.3.jar
- internalSchemas-<platform-version>.jar
- j2objc-annotations-2.8.jar
- jackson-annotations-2.10.3.jar
- jackson-core-2.15.2.jar
- jackson-databind-2.15.2.jar
- jackson-module-jaxb-annotations-2.12.2.jar
- jakarta.activation-api-1.2.1.jar
- jakarta.xml.bind-api-2.3.2.jar
- javassist-3.23.2-GA.jar
- javax.annotation-3.1.1.jar
- javax.enterprise.concurrent-api-1.0.jar
- javax.ejb-3.1.1.jar
- javax.inject-1.jar
- javax.transaction-3.1.1.jar
- jboss-trust-logging- <management-version>-SDK-1.0.jar
- jcip-annotations-1.0-1.jar
- jni-libs.jar
- json-20230227.jar
- json-smart-2.4.11.jar
- log4j-1.2-api-2.20.0.jar
- log4j-api-2.19.0.jar
- log4j-core-2.19.0.jar
- log4j-rolling-appender-20100605-1200-1.2.9.jar
- log4j-slf4j-impl-2.17.2.jar
- metrics-core-3.2.2.jar
- mxparser-1.2.2.jar
- netty-all-4.0.56.Final.jar
- netty-buffer-4.1.4.Final.jar
- netty-codec-4.1.4.Final.jar
- netty-common-4.1.4.Final.jar
- netty-handler-4.1.4.Final.jar
- netty-resolver-4.1.4.Final.jar
- netty-transport-4.1.4.Final.jar
- nimbus-jose-jwt-9.37.jar
- org.apache.oltu.oauth2.client-1.0.1.jar
- org.apache.oltu.oauth2.common-1.0.1.jar
- pgjdbc-ng-0.7-complete.jar
- platformListener-api-<platform-version>.jar

- platformListener-api-impl-<platform-version>.jar
- platformResources-<platform-version>.jar
- postgresql-9.4.1212.jar
- reflections-0.9.9-RC1.jar
- schemas-<platform-version>.jar
- sip-common.jar
- smc-logging-helper-<platform-version>.jar
- snapin-alarms-jaxb-<platform-version>.jar
- snappy-java-1.1.2.6.jar
- speech-search-query-<platform-version>.jar
- spring-aop-4.3.20.RELEASE.jar
- spring-beans-4.3.20.RELEASE.jar
- spring-context-4.3.20.RELEASE.jar
- spring-core-4.3.20.RELEASE.jar
- spring-expression-4.3.20.RELEASE.jar
- spring-jdbc-4. 3.20.RELEASE.jar
- spring-tx-4. 3.20.RELEASE.jar
- srAgentProxy-<platform-version>.jar
- tmclient- <management-version>-SDK-1.0.jar
- xml-apis-1.0.b2.jar
- xmlpull-1.1.3.1.jar
- xstream-1.4.20.jar
- zephyrDataAPI-<platform-version>.jar
- zephyrDataApiFactory-<platform-version>.jar
- zephyrDM-<platform-version>.jar
- zephyrEncryptDecrypt-<platform-version>.jar
- zephyr-logging-client-<platform-version>.jar
- zephyrGlobalData-<platform-version>.jar
- zephyrSecurityManager-<platform-version>.jar
- zephyrUtilities-<platform-version>.jar

JARs in lib/ce_shared

- activemq-all-5.16.7.jar
- amclientsdk-9.5.4_RTM.jar
- authorization_helper-api-<platform-version>.jar
- authorization_helper-impl-<platform-version>.jar
- collaborationBusAPI-<platform-version>.jar
- collaborationBusCore-<platform-version>.jar
- collaborationBusFactory-<platform-version>.jar
- collaborationBusListener-<platform-version>.jar
- commons-logging-1.2.jar
- cr-api-<platform-version>.jar

- cr-api-impl-<platform-version>.jar
- datagrid-api-<platform-version>.jar
- dcm-was.jar
- eac-<platform-version>.jar
- email-api-<platform-version>.jar
- email-api-impl-<platform-version>.jar
- eventing-api-<platform-version>.jar
- eventing-api-impl-<platform-version>.jar
- eventPublisherHelpers-<platform-version>.jar
- http-api-<platform-version>.jar
- http-api-impl-<platform-version>.jar
- internalmessaging-api-<platform-version>.jar
- internalmessaging-api-impl-<platform-version>.jar
- java-facade-<platform-version>.jar
- log4j-web-2.19.0.jar
- logging-api-<platform-version>.jar
- logging-api-impl-<platform-version>.jar
- logging-was-puinfo-<platform-version>.jar
- openssoclientsdk-9.5.4_RTM.jar
- opensso-sharedlib-9.5.4_RTM.jar
- p2p-<platform-version>.jar
- pfaServices-<platform-version>.jar
- ports-api-<platform-version>.jar
- ports-api-impl-<platform-version>.jar
- reliableeventing-api-<platform-version>.jar
- reliableeventing-api-impl-<platform-version>.jar
- resource-datastore-<platform-version>.jar
- schedConf-api-<platform-version>.jar
- schedConf-api-impl-<platform-version>.jar
- serviceActivity-api-<platform-version>.jar
- serviceActivity-api-impl-<platform-version>.jar
- smc-api-<platform-version>.jar
- smc-api-impl-<platform-version>.jar
- smc-call-reconstruction-<platform-version>.jar
- sms-api-<platform-version>.jar
- sms-api-impl-<platform-version>.jar
- ssal-common-<platform-version>.jar
- ssal-well-behaved-<platform-version>.jar
- ssl_util-api-<platform-version>.jar
- ssl_util-api-impl-<platform-version>.jar
- systemState-api-<platform-version>.jar
- systemState-api-impl-<platform-version>.jar

- zangconnectionlib-<platform-version>.jar
- zephyrDataUtil-<platform-version>.jar
- zephyrHelper-<platform-version>.jar

JARs in lib/ce_isolated

- Breeze-AAMSConnectorAPI-<platform-version>.jar
- ceSystemStatus-<platform-version>.jar
- commons-codec-1.11.jar
- httpclient-4.5.14.jar
- httpcore-4.4.16.jar
- javax.mail-1.6.0.jar
- p2p-singleton-<platform-version>.jar
- slf4j-api-1.7.26.jar
- slf4j-log4j12-1.7.26.jar

Jars provided by IBM Websphere

For the list of jars/libraries provided by IBM Websphere that are running on Avaya Breeze® platform, see
https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/opensourcesoftwareapis.html

Parent First Versus Parent Last Classloading

Avaya Breeze® platform only supports parent first classloading for snap-ins. Therefore, use of any mechanism that results in parent last classloading for a snap-in is not supported. For example, it is possible to place a deployment.xml file in the EAR portion of a snap-in as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<appdeployment:Deployment xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:appdeployment="http://www.ibm.com/websphere/appserver/schemas/5.0/
appdeployment.xmi" xmi:id="Deployment_1471449536828">
   <deployedObject xmi:type="appdeployment:ApplicationDeployment"
xmi:id="ApplicationDeployment_1471449536829" startingWeight="10">
      <modules xmi:type="appdeployment:WebModuleDeployment"
xmi:id="WebModuleDeployment_14714495368300" startingWeight="10000" uri="TestBreeze.war"
classloaderMode="PARENT_LAST"/>
     <classloader xmi:id="Classloader_1471449536831" mode="PARENT_LAST"/>

     </deployedObject>

</appdeployment:Deployment>
```

Use of this file with a mode of PARENT_LAST results in an unsupported configuration for your snap-in.

Next steps

Congratulations! You have created a service that is ready to be installed on System Manager. To learn how to install on the System Manager and for more information on the process of installing, configuring and testing a service, see *Deploying Avaya Breeze® platform* and *Administering Avaya Breeze® platform*. The rest of this guide focuses on details of service development and the capabilities and other aspects of the collaboration API. There are also many sample services on the DevConnect website that show how to write different types of applications such as HTTP services and HTTP REST.

Chapter 3: Developing the service

You can deploy services using System Manager or Eclipse plug-ins.

Developer update method

Avaya Breeze® platform is designed to install services from a central server, the System Manager, across the network to multiple Avaya Breeze® platform servers. Consequently, movement of a service to an Avaya Breeze® platform server is subject to network delays and outages. This characteristic prevents the rapid updating of services as is desirable when developing and debugging services. In addition, installation from the System Manager requires that the service version be incremented each time. So, to make it easier for developers, the developer update method is an alternative to get an updated service up and running. It bypasses the System Manager and runs the service on the Avaya Breeze® platform server quickly and without the need of updating the version number each time.

Important:

This method only works on subsequent installs of the service. The initial install of a particular service must be done on the System Manager.

Updating a service

Before you begin

This method only works on services that have already been installed on the System Manager.

Procedure

1. Log in to the Avaya Breeze® platform server.
2. Run deploy_service –l to list the services already installed. If your service is not there, go the System Manager and install it there.
3. Copy the svar file for your service to the /tmp directory on the Avaya Breeze® platform server.

   If you develop in a linux environment, you can use the **scp** command to copy the file from your development system to the Avaya Breeze® platform. If you develop in a Windows environment, you could use cygwin tools, PSCP (a Putty tool), or similar to copy the file.

4. Run deploy_service –d /tmp/filename.svar to quickly install your service.

   Note:

   If you change the properties.xml, you must update the service version, delete and install on the System Manager again before you can use the developer update method. If you change the version number of the service, you must install via System Manager. See the section on Service Versioning for more information.

Service versioning

Introduction

Once you have put the initial version of your service into production, you are already thinking about the improvements to make for the next version. Avaya Breeze® platform can run and manage multiple versions of the same service at the same time. Once your service goes into production, the administrators will spend time and effort integrating your service into Avaya Breeze® platform by configuring the service's attributes and adding the service into service profiles. If you keep the version number the same, delete the old version and load the new service, that effort will have to be duplicated. Service versioning allows you to preserve the configuration of your service when a new version is loaded. It also allows the use of two or more different versions of the same service at the same time. For example, the Accounting Department may use a different version of a service in their service profile than the Marketing Department uses.

Instead of new and old, we use the terms earlier, later and latest for versions of a service. The version number determines what is considered a later version or an earlier version of the service, not when it was built. To produce a later version of your service, you simply update the serviceVersion property in your service pom.xml from 1.0.0.0.0 to 1.1.0.0.0, as shown below. If you are using eclipse, you can find this pom.xml file directly under the project that has just your service name without an svar, ear or war extension.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/

XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.mycompany</groupId>

<artifactId>testservice</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<properties>
<serviceName>TestService</serviceName>
<serviceVersion>1.1.0.0.0</serviceVersion>
</properties>

<modules>
<module>testservice-war</module>
<module>testservice-ear</module>
<module>testservice-svar</module>
</modules>
```

```
</project>
```

The new version number should be larger than the earlier version (see below). Now when you build your service it will be a later version of the service. You can load and install a later version of the service without deleting the earlier version and the entire configuration for the service will be retained. The service attribute values are shared amongst all the versions, so there is no need to reconfigure them. You should avoid deleting the last remaining version of a service. The only times you would do that is if you are forever done with that service and have no intention of ever loading any version of it again or the service is under development and you need to delete it to make changes. When you delete the lone remaining version of a service, its attributes values will be removed, and the service will be removed from any service profiles that contain it. The service must be configured again when it is loaded.

Service version numbering considerations

The service version number consists of 5 numbers separated by 4 periods. Some valid version numbers are 1.0.0.0.0, 2.0.0.0.1, 2.220.87.200.20 and 888.999.333.444.444. Some invalid ones are 1.2.3 (less than 5 numbers), 1.2.3.4.5.6 (more than 5 numbers) and 1.0.0.0.revA (revA is not a number). The numbers can be very large, but we recommend using 5 digits or less. The later version is the version with the larger version number determined by comparison as in the following example. Version A is x1.x2.x3.x4.x5 and version B is x6.x7.x8.x9.x10 where x1 - x10 represent arbitrary numbers. Start by comparing the left most numbers of each version, x1 and x6. If x1 is greater than x6, then Version A is larger. If x1 is less than x6, then Version B is larger. If x1 and x6 are equal then compare the next number to the right in each version, x2 and x7. Repeat the comparison and move to the right if equal until the larger version is determined. So, version 2.0.0.0.0 is later than 1.9999.1.1.1 and 2.0.0.1.0 is later that 2.0.0.0.60. Beware of using leading zeros. They are insignificant in the comparison. So, 1.005.077.000.01, 01.0005.77.0.00001 and 1.5.77.0.1 are all equal. The latest version is the version with the largest version number. Version numbers between different services are not related. So, one cannot tell whether TestService 3.0.0.0.0 is later or earlier than HelloService 2.0.0.0.0.

Attribute considerations

When developing a later version of a service, your later version may need to define new attributes. That is fine; just add the new ones into the properties.xml file as you did for the earlier version. However, you cannot modify anything about the attributes that existed in an earlier version. This rule is to make sure the earlier version can still operate in its original configuration. Failure to observe this rule could cause an error when you try to load your later service. If you need to change something about an existing attribute in a later version, create a new attribute and have the later version use the new attribute.

Properties.xml considerations

The properties.xml contains other information about the service version. The properties.xml is located under the .svar module in src/main/resources. If you need to update information contained in the properties.xml, you must update the version number or delete the service before loading and installing again. If you delete the service rather than update the version, and no other versions of the service are loaded, you will lose any NON-DEFAULT attribute values and the service will be removed from any service profiles that contain it as described above.

Service Profile considerations

There are three ways than an administrator can specify the version of a service that is used in a particular service profile: specific version, latest, or preferred.

- If a specific version is specified, that version of the service will always be invoked for users with the given service profile regardless of what other versions of that service are installed.
- If “latest” has been specified, the latest version of the service will be invoked for users with the given service profile. If a newer version of the service is installed, that newly installed version will be invoked without any further actions by the administrator.
- If “preferred” has been specified, then the designated “preferred” version of the service will be invoked for users with the given service profile. This is similar to the specific version setting but has the added benefit that the preferred version is applicable to both calls and HTTP messages. If administrators want to change the version of the service being invoked for both calls and HTTP, they can do so by updating the preferred version in a single place.

The *Administering Avaya Breeze® platform guide* explains about:

- How to assign services to service profiles
- How to designate a service version as being the preferred version.

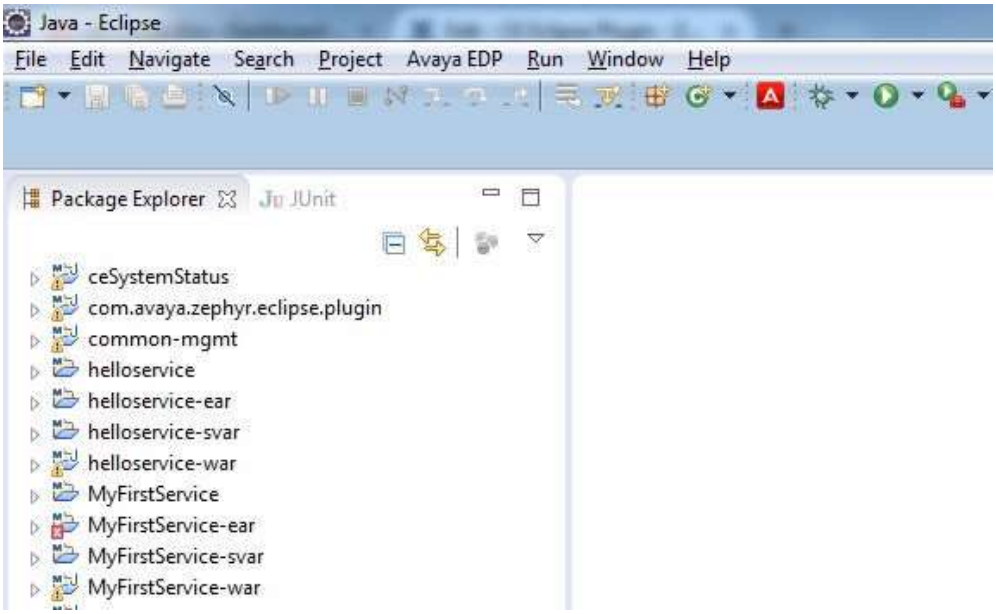
Configuring the Eclipse plug-in

About this task

Using the Eclipse plug-in, developers can manage System Manager and Avaya Breeze® platform servers from the Eclipse IDE. The Eclipse plug-in is included in the Avaya Breeze® platform SDK.

Before you begin

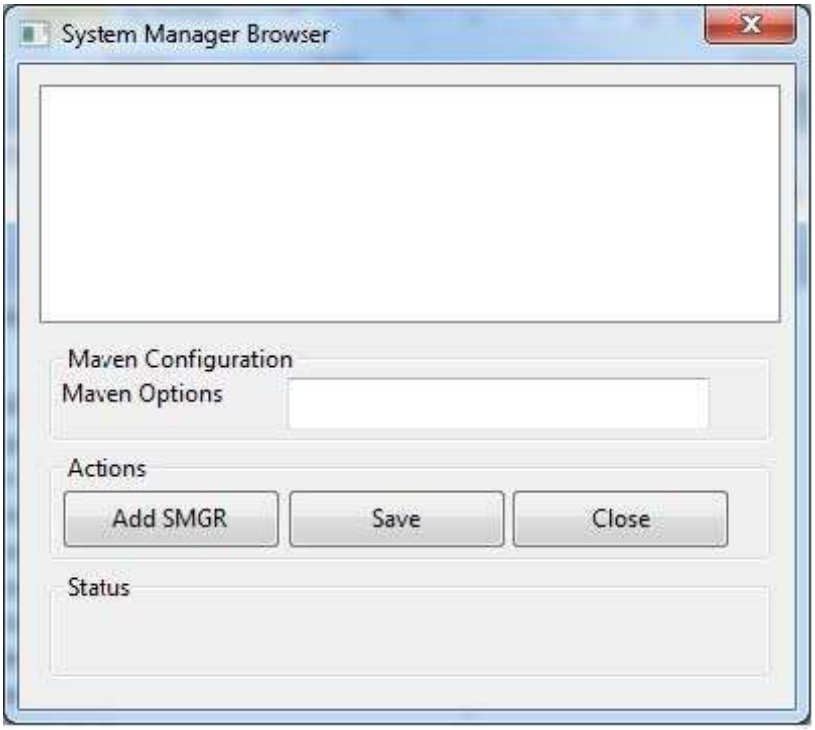
Locate the Eclipse plug-in icon.



Procedure

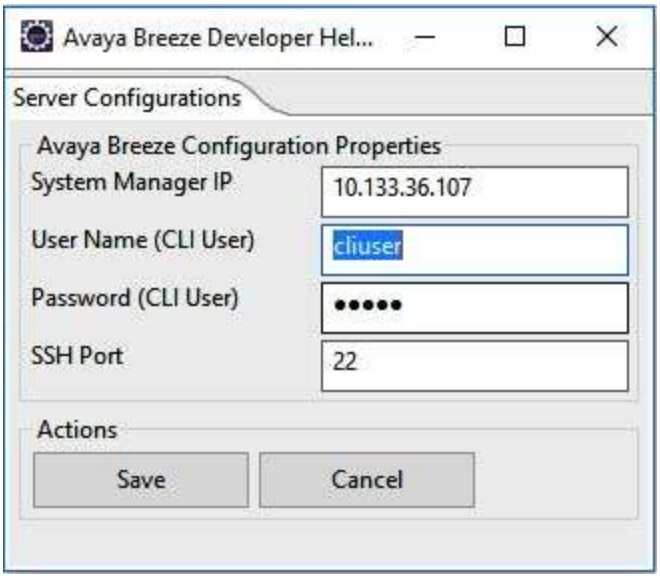
1. Click the Avaya icon on the tool bar.

   The system displays the System Manager Browser window.



2. Click **Add SMGR**.

   The system displays the Server Configurations tab of the Avaya Breeze Developer Helper window.



3. Configure the following fields for a CLI user and click **Save**.
   - **System Manager IP**
   - **User Name (cli user)**
   - **Password (cli user)**
   - **SSH Port**: The default SSH port is the 22 TCP port.

SSH is a network protocol that supports a secure connection to remote computers for administrators to execute commands. The SSH port is the port on which the SSH server listens to the remote server.

4. Click **Save**.

   The system displays the cluster tree of all the Avaya Breeze® platform and System Manager servers, along with the default configuration details.

   - A green status indicates that the Avaya Breeze® platform and System Manager servers are connected and are correctly configured.
   - A red status indicates that the servers are not connected or not correctly configured.

5. **(Optional)** To change the Avaya Breeze® platform or System Manager server configuration, select the server and right-click the server name.

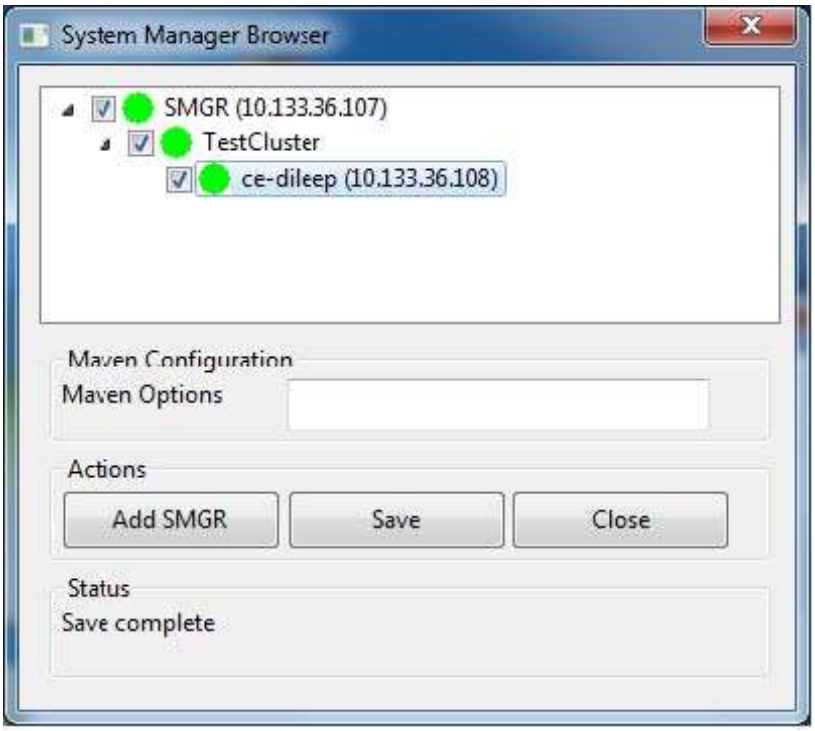
   The system displays the options to change the server configuration.

6. Click **Edit System**.

   The system displays the Server Configurations tab.

7. Change the relevant configuration and click **Save**.

   If the changed configuration is correct, the system changes the status of the server to green.

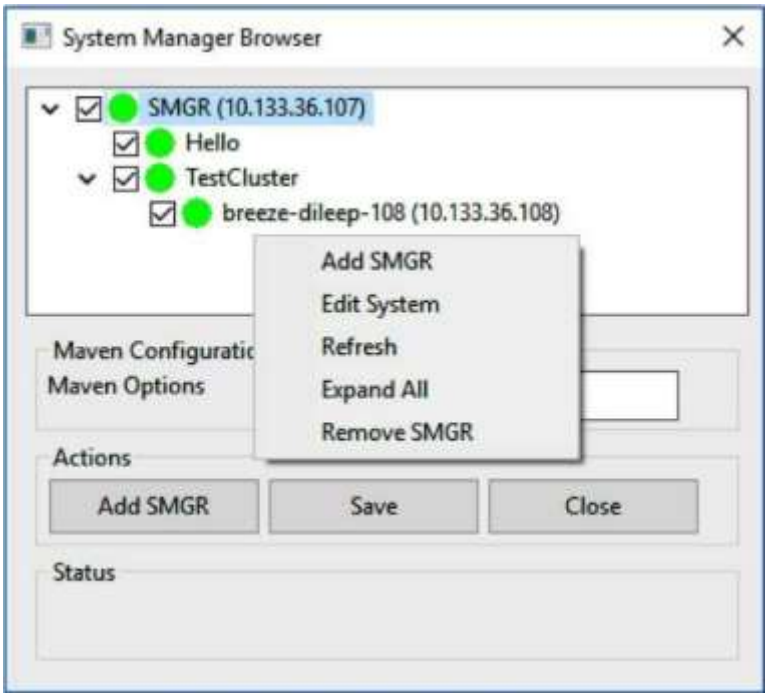


Next steps

Repeat this procedure for all the System Manager and Avaya Breeze® platform servers that have the red status.

System Manager Browser field descriptions

System Manager Browser supports the following administration for Avaya Breeze® platform and System Manager:



Name	Description
Add SMGR	Add a System Manager node in System Manager Browser.
Edit System	Edit the Avaya Breeze® platform and System Manager server configurations. System Manager Browser displays an Avaya Breeze Developer Helper window to edit the server configuration.
Refresh	Apply the changes made to the Avaya Breeze® platform and System Manager servers. All Avaya Breeze® platform and System Manager selections are lost after you apply the new changes. You must select the Avaya Breeze® platform nodes and clusters again.
Expand All	Expand the Avaya Breeze® platform and System Manager nodes tree.
Remove SMGR	Remove selected System Manager nodes.

Using the Eclipse IDE

About this task

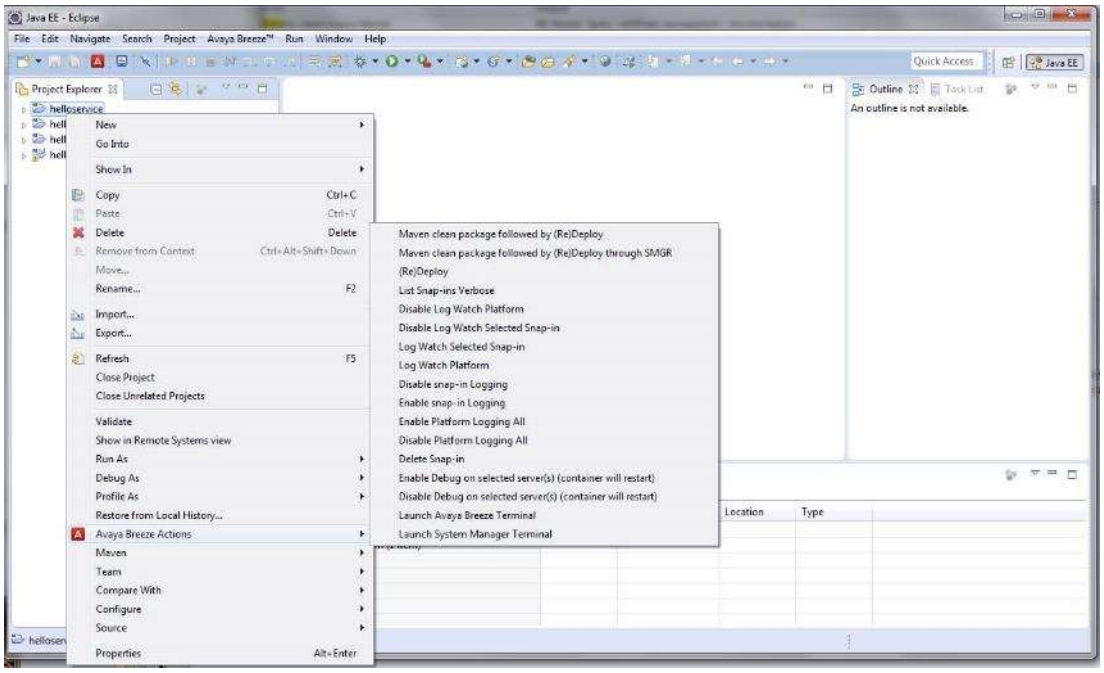
The Maven CLI options, such as -DskipTests=true, -debug, or -quiet, must be separated by a comma.

Before you begin

- Configure the Eclipse plug-in.
- Ensure that all the System Manager and Avaya Breeze® platform servers and clusters are connected to the Eclipse plug-in.

Procedure

1. On the Eclipse ID, right-click the Avaya Breeze® project.



The Eclipse IDE displays the available project options.

2. Click **Avaya Breeze® Actions**. The Eclipse IDE displays the Avaya Breeze® project options.

Actions supported for the project

The following actions are supported for a project.

Deploy/Redeploy

This action will automatically transfer the svar file built using Maven to either the System Manager or Avaya Breeze® platform based on the selected action. The first installation of the service will be done through System Manager and all the subsequent deployment

(redeploy) will be executed directly on Avaya Breeze® platform. The progress of the action can be viewed through the Eclipse console window (**Window** > **Show View** > **Console**).

Redeploy through System Manager

This action will clean build the service and deploy the svar through System Manager.

List Snap-ins Verbose

This action will list the services deployed on Avaya Breeze® platform server. The progress of the action can be viewed through the eclipse console window (**Window** > **Show View** > **Console**).

Disable Platform Logging All

This action will stop tailing logs on console that is performed by action Log Watch All.

Disable Log Watch Selected Snap-in

This action will stop tailing of logs on console performed by action Log Watch Selected Service.

Log Watch Selected Snap-in

The selected service specific logs can be viewed by executing this command. Logs will get automatically updated on the console window.

Log Watch Platform

All the Avaya Breeze® platform logs can be viewed by executing this command. The will be automatically updated on the console window.

Enable Platform Logging All

This is to set the logging level of Avaya Breeze® platform server to All. Set the logging level of Avaya Breeze® platform server as info.

Delete Snap-in

This action will delete the selected service from the system. The progress of the action can be viewed through the eclipse console window (**Window** > **Show View** > **Console**).

Enable Debug on Selected server(s) (container will restart)

This action will put Avaya Breeze® platform servers in debug mode to help service developers to debug the deployed/installed services.

Disable Debug on Selected server(s) (container will restart)

This action will put selected Avaya Breeze® platform servers out of debug mode.

Launch Avaya Breeze® platform Terminal

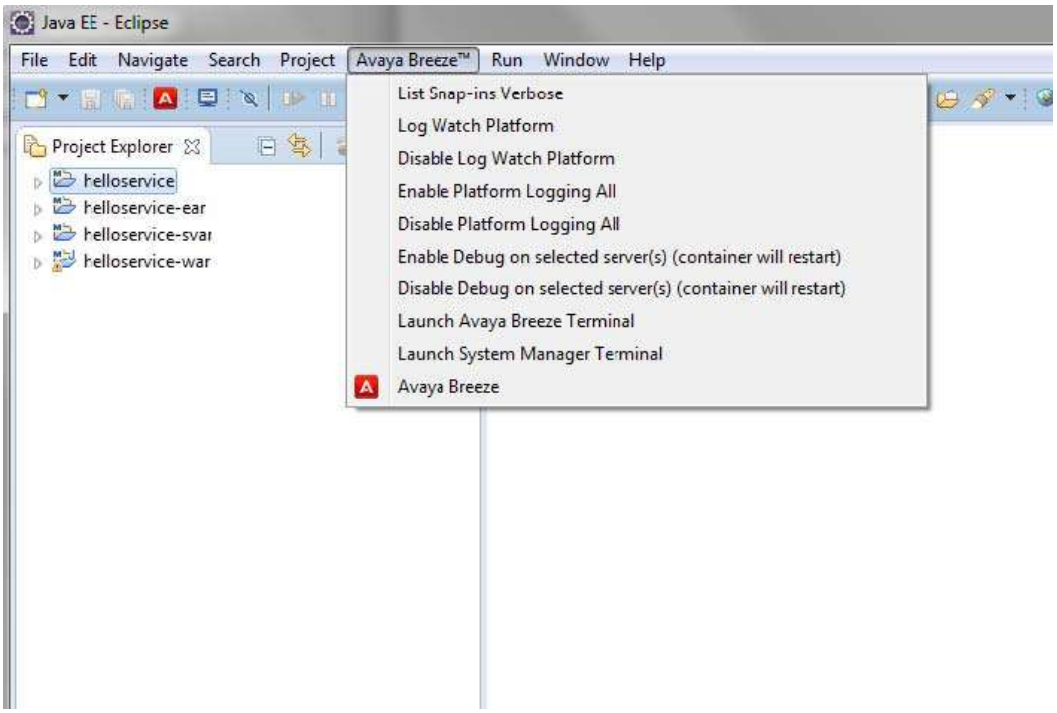
Launch the Avaya Breeze® platform terminal by using the configured user name and password.

Launch System Manager Terminal

Launch the System Manager terminal by using the configured user name and password.

Avaya Breeze® Global actions

The following global actions are supported for Avaya Breeze® platform.



List Snap-ins Verbose	This action will list the services deployed on Avaya Breeze® platform server.
Log Watch Platform	The Avaya Breeze® platform logs can be viewed by executing this command.
Disable Log Watch Platform	This action will stop tailing logs on console that is performed by action Log Watch Platform.
Enable Platform Logging All	This is to set the logging level of Avaya Breeze® platform server to ALL (i.e., enable fine, finer and finest logging levels). Default logging level for Avaya Breeze® platform server set as INFO.
Disable Platform Logging All	This action will stop tailing logs on console that is performed by action Log Watch Platform.
Enable Debug on Selected server(s) (container will restart)	This action will put Avaya Breeze® platform servers in debug mode to help. Snap-in developers to debug the deployed/installed services.
Disable Debug on Selected server(s) (container will restart)	This action will put selected Avaya Breeze® platform servers out of debug mode.
Launch Avaya Breeze® Terminal	Launch the Avaya Breeze® platform terminal by using the configured user name and password.

Launch System Manager Terminal	Launch the System Manager terminal by using the configured user name and password.

Enabling remote debugging on Avaya Breeze®

About this task

The default debugging port is not open for remote debugging by default because of the security risks associated with unauthorized access to the enterprise network. It might be feasible to open the debugging port for remote debugging only in lab-based environments.

To perform an indepth debugging, install a remote debugger, such as the Eclipse plug-in for Avaya Breeze in order to step through your snap-in's code. There are 2 easy mechanisms to do so.

- Invoke the "Enable Debug on Selected server(s)" action in Eclipse (This is only available if the Breeze Eclipse Plugin is installed).
- Run the "enableDebugCE" command on the Breeze CLI.

Procedure

1. To enable a port for remote debugging, do one of the following:
   - In Eclipse, invoke the Enable Debug on Selected server(s) action.

     This action is available only if you installed the Avaya Breeze® Eclipse plug-in.

   - On the Avaya Breeze® CLI, run the following command: enableDebugCE.
2. To disable the debug port, do one of the following:
   - In Eclipse, invoke the "Disable Debug on Selected server(s)" action.
   - On the Avaya Breeze® CLI, run the following command: disableDebugCE

Result

- Port 8787 is opened on the Avaya Breeze® firewall.
- Websphere shuts down and all Avaya Breeze® snap-ins stop.
- Websphere restarts having bound its debugger port to port 8787 on the "Management NIC".

There are no negative ramifications to leaving debug mode on all the time in the lab. You would not want to do this on production systems due to the security risk (incoming connections to that port are not authenticated in any way).

Converting an older project to Avaya Breeze® platform 3.9

Before you begin

- 3.9 SDK is successfully installed.
- JDK 1.8 is installed.

Procedure

1. In the pom.xml present in war project replace dependency below :

```
<dependency>
                <groupId>com.avaya.collaboration.api</groupId>
                   <artifactId>avaya-aura-collaboration-api-x.x</artifactId>
                   <version>x.x.x.x.x</version>
                   <scope>provided</scope>
          </dependency>
```

With

```
<dependency>
                <groupId>com.avaya.collaboration.api</groupId>
                   <artifactId>avaya-aura-collaboration-api-3.9</artifactId>
                   <version>3.9.0.0.xxxxx</version>
                   <scope>provided</scope>
</dependency>
```

2. In the src/main/resources/CARRule.xml file under the war project, add the following rule:

```
<TerminatingServiceRule desc="Interested in serviceName featureURI as named app">
                <FeatureURI>serviceName</FeatureURI>

      </TerminatingServiceRule>
```

3. Replace the JRE system Library 1.x dependency with 1.8 JRE System library.

4. In the pom.xml for service svar project , make changes below:

   a. Replace the highlighted part:

```
artifactItem>
    <groupId>com.avaya.collaboration.api</groupId>
    <artifactId>avaya-aura-collaboration-api-x.x</artifactId>
    <version>x.x.x.x.x</version>
    <type>jar</type>
    <overWrite>true</overWrite>
    <outputDirectory>${project.build.directory}/tmp</outputDirectory>
    <destFileName>sdk.properties</destFileName>
          <includes>META-INF/MANIFEST.MF</includes></artifactItem>
```

With

<artifactId>avaya-aura-collaboration-api-3.9</ artifactId>

<version>3.9.0.0.xxxxxx**</version>**

b. Replace the highlighted part:

```
<dependency>
        <groupId>com.avaya.zephyr.zephyrCommon.xsds</groupId
                >
        <artifactId>archive-xsd</artifactId>
        <version>x.x.x.x.x</version>
        <exclusions>
        <exclusion>
        <artifactId>smc-api</artifactId>
        <groupId>com.avaya.zephyr.services.smc</groupId>
        </exclusion>
        </exclusions>
    </dependency>
```

With

<version>3.9.0.0.xxxxxx**</version>**

5. Add the following snippet:

```
<validationSet>
        <dir>${basedir}</dir>
        <systemId>${basedir}/target/dependency/alarms.xsd</systemId>
        <includes>
        <include>src/main/resources/alarms.xml</include>
        </includes>
</validationSet>
```

After having added this snippet, the developer will be able to specify service-specific alarms in the following file: <*servicename*>-svar/src/main/resources/alarms.xml.

6. Add the following snippet to dist.xml located at /<svcname>-svar/src/main/assembly/dist.xml:

```
<file>
        <source>src/main/resources/alarms.xml</source>
        <outputDirectory>/</outputDirectory>
        <filtered>true</filtered>
 </file>
```

Snap-in start/stop from Avaya Breeze® Service Management page

You can start and stop a snap-in for selected clusters from the Avaya Breeze® Service Management page. You can use the **Start** and **Stop** buttons on the Avaya Breeze® Service Management page to start or stop snap-ins. For additional information, see *Administering Avaya Breeze® platform*.

Creating a snap-in with the start/stop functionality

Procedure

1. Modify the properties.xml of the service to enable the start/stop options by adding the following entry and build the svar:

```
<attribute name="SnapInStoppableName">
    <displayName>EnableStartStopForSnapin</displayName>
    <helpInfo>Enable Start Stop For Snapin</helpInfo>
    <validation name="anyString">
            <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <factory>
            <value>true</value>
            <user_changeable>true</user_changeable>
    </factory>
</attribute>
```

2. This snap-in can be installed in a started state or a stopped state based on value of tag pattern.
   - When value is set to *false*, the snap-in is installed in a Stopped state on the cluster.
   - When value is set to *true*, the snap-in is installed in a Started state on the cluster.

```
<attribute name="StartOnInstall">
    <displayName>StartOnInstall</displayName>
    <helpInfo>This attribute will enable Snap-In Stop-Start capabilities. This cannot be
    changed from UI.
    </helpInfo>
    <validation name="booleanType">
    <type>STRING</type>
    <pattern>false.</pattern>
    </validation>
    <admin_visible>false</admin_visible>
    <admin_changeable>false</admin_changeable>
    <factory>
        <value>true</value>
```

```
                <user_changeable>false</user_changeable>
            </factory>
        </attribute>
```

Note : Above steps are all that is needed to enable start/stop functionality for a snap-in.

Configuring log file size

As is explained in the Logger section below, log statements from services are written to a log file specific to that service. Developers of services can specify their desired disk space for log files. This is done by adding the highlighted tag to the properties.xml file. The default is 10 MegaBytes. Please note that this line is commented out by default. The comment tags will have to be removed in order to have a modified value take effect.

```
<?xml version="1.0" encoding="UTF-8"?>

<service xmlns="http://archiveschemas.aus.avaya.com/properties"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://archiveschemas.aus.avaya.com/properties
properties.xsd"

name="${serviceName}" version="${serviceVersion}" application="${serviceName}
${serviceVersion}">

        <smgr>
                <description>Test Service</description>
                <term_order>1</term_order>
                <term_group>1</term_group>
                <fs_component>true</fs_component>
        </smgr>
        <log_space>10MB</log_space>
</service>
```

Service attributes

Attribute definition and access levels

A snap-in can optionally define one or more configurable snap-in attributes. The snap-in can then query the values during processing. Snap-in attributes can be administered and retrieved in three different levels. The getAttribute() method has two forms. The first form specifies a user and an attribute name, the second specifies only an attribute name. If the first form is used, all three levels are searched in order. If the second form is used, only levels 2 and 3 are searched. Here are the levels:

1. Service Profile values are administered by service profiles which are assigned to users. If the specified user is found and the specified attribute value is found in the Service

Profile of that user, then that value is returned. If not, the attribute is searched for in the next level. This is the only level that has values by user.
2. Service Cluster values are administered by cluster. The search for the attribute name is made on the cluster where the snap-in is running. If a value is found on the cluster, that value is returned. If not, the attribute is searched for in the next level.
3. Service Global values are assigned a default value by the snap-in writer. Each default value may be overridden by administration. If the value was overridden, then the overridden value is returned else the default value is returned.

As an example, consider a service that would log calls to a certain user to a database. This service could have 1 attribute that enables or disables the feature on a service profile basis. Another attribute could be the address of the server that hosts the database – this attribute could be set for certain clusters and the service global value could be used for any clusters that do not specifically define a database server. You might even have attributes that have only Service Cluster or Service Global values

The attributes that a service uses are declared in the properties.xml file inside the SVAR. In a project generated by the service archetype, the properties.xml file can be found in the SVAR module in src/main/resources/properties.xml. The following is an example of an attribute declared in properties.xml:

```
<smgr>
...
<attribute name="displayString">
        <displayName>Display String</displayName>
        <helpInfo>String used for caller's caller ID display</helpInfo>
        <validation name="anyString">
                <type>STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <factory>
                <value>Hello from Avaya Breeze</value>
                <user_changeable>true</user_changeable>
        </factory>
</attribute>
...
</smgr>
```

The definition of the attribute includes the following parts:

- Attribute name: the name that will be used to reference the attribute in your code.
- Display name: the name that will be used for this attribute in the Service Profile editor.
- Help Info: descriptive text for this attribute in the service profile editor.
- Type: Defined in the following “Avaya Breeze® attribute types” section.
- Admin visible: Determines whether the attribute appears in the System Manager administration UI.
- Factory: this section describes the default values for the attribute

Note:

The order in which the elements of an attribute are mentioned in its definition is important for the snap-in to be built successfully.

Avaya Breeze® platform attribute types

Snap-in can declare attributes with any of the following types and avail the in-built validation capabilities.

- String – any String value that is displayed as a text box on the user interface (UI).

```
<attribute name="stringAttribute">
        <displayName>StringAttribute</displayName>
        <helpInfo>Example of String Attribute</helpInfo>
        <type>STRING</type>
        <validation name="anyString">
            <type>STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <factory>
                <value>sample</value>
                <user_changeable>true</user_changeable>
        </factory>
</attribute>
```

- Integer – Any integer value within the range -2147483648 to 2147483647 that is displayed as a text box.

```
<attribute name="integerAttribute">
        <displayName>IntegerAttribute</displayName>
        <helpInfo>Example of Integer Attribute</helpInfo>
        <type>INTEGER</type>
        <encrypted>false</encrypted>
        <validation name="anyInteger">
            <type>STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <factory>
                <value>80</value>
                <user_changeable>true</user_changeable>
        </factory>
</attribute>
```

- Encrypted Integer – Any value belonging to type integer that will be encrypted once the snap-in is loaded. This is done by setting the encrypted tag to true. Encrypted Integer is displayed as a text box. The value is rendered in an encrypted format on the UI.

```
<attribute name="encryptedIntegerAttribute">
    <displayName>EncryptedIntegerAttribute</displayName>
    <helpInfo>Example of Encrypted Integer Attribute</helpInfo>
```

```
    <type>INTEGER</type>
    <encrypted>true</encrypted>
    <validation name="anyInteger">
        <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <factory>
        <value>65</value>
        <user_changeable>true</user_changeable>
    </factory>
</attribute>
```

- Range – Validation supported to check if the value entered by the user falls within a specific range of integers. The validation check is performed on when you click Commit after changing the attribute value on the Attribute page in System Manager.

```
<attribute name="rangeAttribute">
    <displayName>RangeAttribute</displayName>
    <helpInfo>Example of Range Attribute</helpInfo>
    <type>RANGE</type>
    <valueChoices>200-700</valueChoices>
    <validation name="anyInteger">
        <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <factory>
     <value>500</value>
            <user_changeable>true</user_changeabe>
    </factory>
</attribute>
```

- Boolean – Value of this attribute type, by default, can be set to true or false. The attribute is rendered as a check box on the Snap-in Attributes page.

```
<attribute name="booleanAttribute">
    <displayName>BooleanAttribute</displayName>
    <helpInfo>Example of Boolean Attribute</helpInfo>
    <type>BOOLEAN</type>
    <validation name="anyBoolean">
        <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <factory>
        <value>true</value>
        <user_changeable>true</user_changeable>
    </factory>
</attribute>
```

- Choice – A Snap-in developer can restrict the value of an attribute by defining the attribute type as Choice by providing a comma-separated list of values that are not editable. The values are rendered in a drop-down format on the Attributes page. A user can select only a single value from the list of values provided in the choice drop-down list.

```
<attribute name="choiceAttribute">
        <displayName>ChoiceAttribute</displayName>
        <helpInfo>Example of Choice Attribute</helpInfo>
        <type>CHOICE</type>
        <valueChoices>value1,value2,value3</valueChoices>
        <validation name="anyChoice">
                <type>STRING</type>

        </validation>
        <admin_visible>true</admin_visible>
        <factory>
                <value>value2</value>

                <user_changeable>true</user_changeable>

        </factory>
</attribute>
```

- List – A comma separated list of values, in which you can add or delete elements. A list is rendered on the UI in separate text boxes with buttons to add or delete a value. A user is able to add more values to this list by clicking on the + button that is present next to the last value in the list. A user can also delete a value in this list by clicking on the - button that is present next to each of the values added previously. Users might add the values manually or the values might be added by default, owing to the factor default values shown in the following snippet:

```
<attribute name="listAttribute">
        <displayName>ListAttribute</displayName>
        <helpInfo>Example of List Attribute</helpInfo>
        <type>LIST</type>
        <validation name="anyList">
                <type>STRING</type>

  </validation>
  <admin_visible>true</admin_visible>
  <factory>
                <value>Value1,Value2,Value3</value>

                <user_changeable>true</user_changeable>

        </factory>
```

- </attribute> Cluster – Snap-in developers can define a new attribute with the Cluster type to select an Avaya Breeze cluster name. All available Avaya Breeze cluster names are displayed in a drop-down format. You can select a cluster name as the value. This attribute type is useful when a snap-in is installed on multiple clusters with one of the

clusters being the main cluster for the snap-in. You can set the value of the new attribute to point to the main cluster.

Example:

```
<attribute name="clusterAttribute">
        <displayName>ClusterAttribute</displayName>
        <helpInfo>Example of Cluster Attribute</helpInfo>
                <type>CLUSTER</type>
        <validation name="anyCluster">
                <type>STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <factory>
                <value>clusterName</value>
                <user_changeable>true</user_changeable>
        </factory>
</attribute>
```

Note:

If you want to change the attribute type in a newer snap-in version, use one of the following methods:

- Uninstall and delete from System manager, all older versions of the snap-in.
- Change the name of the attribute for which type needs to be changed, increment the snap-in version number. Build and load this new snap-in version for use.

Attribute Grouping, Ordering, and Scope

Snap-in developers now have more control over how the snap-in's attributes are displayed on the Attributes Configuration page.

- multiple attributes can be combined into an attribute group. The attributes in an attribute group will be shown together on the Attributes Configuration page.
- the scope of an attribute can be restricted, to control the visibility of the attribute on each of the tabs of the Attributes Configuration page.
- the sorting order of attributes can be specified explicitly.

These features only control how attributes are displayed on the Element Manager's Attribute configuration page. There is no change to the underlying attribute behavior.

An extract from a snap-in's properties.xml file is shown below, demonstrating all the above features. Refer below for more details on each feature.

```
<attribute name="ServerHostname"> <!-- the first three attributes are in the "Server"
attribute group -->
    <group>
    <group_name>Server</group_name> <!-- the group name is used as the title for this group
on the Attributes page -->
    <group_order>1</group_order>
    </group>
```

```
    <scope>Global,Cluster</scope> <!-- these attributes will only be shown on the Global and Service
    Clusters tabs of the Attributes page -->
    <attr_order>1</attr_order>
</attribute>
<attribute
name="ServerUsername">
    <group>
    <group_name>Server</group_name>
    <group_order>1</group_order> <!-- this value is used when sorting multiple attribute groups to
    display on the Attributes page -->
    </group> <scope>Global,Cluster</scope>
    <attr_order>2</attr_order>
</attribute>
<attribute name="ServerPassword">
    <group>
    <group_name>Server</group_name>
    <group_order>1</group_order> </group>
    <scope>Global,Cluster</scope>
    <attr_order>3</attr_order>
</attribute>
<attribute name="SecureConnection"> <!-- the next two are not in an attribute group, and are also
limited to the ServiceProfile scope -->
    <scope>ServiceProfile</scope>
    <attr_order>1</attr_order>
</attribute>
<attribute name="ConnectionTimeout">
    <scope>ServiceProfile</scope>
    <attr_order>2</attr_order>
</attribute>
```

Attribute Grouping

Put attributes into a group by adding a group tag to the attribute definition in the properties.xml file. Attribute groups are shown separately on the snap-in's Attribute Configuration page and be hidden (collapsed) to improve usability when there are many attributes and groups. Attributes not part of an attribute group are placed into a default group and displayed first on the page.

Attribute Scope

An attribute's scope, i.e., which tabs of the Attribute Configuration page the attribute is displayed on, can now be controlled by specifying one of four possible values:

- Global - the attribute will only be shown on the Service Globals tab.
- ServiceProfile - the attribute will be shown only on the Service Profiles tab.
- Global,Cluster - the attribute will be shown on the Service Globals and Service Clusters tab
- Global,Cluster,ServiceProfile - the attribute will be shown on all three tabs of the Attribute Configuration page.

If no value is provided, the default value is Global,Cluster,ServiceProfile.

Attribute Order

Attributes are currently displayed on the Attribute Configuration page in alphabetical order. Snap-in developers can now control the order of attributes by providing a value for attr-order for each attribute. If an order is not specified, the attributes are still sorted alphabetically.

If there are multiple attribute groups, the order of the attribute groups can also be specified by specifying the group_order for each group.

Attribute Value Validation using Regular Expressions

Regular expressions provide a way for snap-in developers to validate attribute values. Snap-in developers can provide a regular expression for each attribute in the snap-in's properties.xml file. The value entered by the administrator is matched against the regular expression, and if they do not match, an error message is displayed, and the commit is blocked. Snap-in developers should include a user-friendly description of the valid attribute values which will match the regular expression.

The regular expression is specified in the validation section in the properties.xml file. A couple of examples are given below.

```
<attribute name="Extension">
    <displayName>User's Extension</displayName>
    <helpInfo>The user's extension (should be a numeric value with at least 4 digits).</ helpInfo>
    <validation name="Numeric">
    <type>STRING</type>
    <pattern>[1-9][0-9]{3,}</pattern>
    </validation>
</attribute>
<attribute name="Username">
    <displayName>name</displayName>
    <helpInfo>The username. Valid usernames start with an alphabet and can end with one or more
    digits.</helpInfo>
    <validation name="Alphanumeric">
    <type>STRING</type>
    <pattern>[a-z][a-z]*[0-9]*</pattern>
    </validation>
</attribute>
```

Override the factory default values of attributes for snap-ins

Snap-in developers can now override the factory default values of attributes for snap-ins that are loaded on System Manager. Use the attribute_overrides XML element to define the new factory default values for a set of snap-in attributes to a value specified in the attribute_override element. Each attribute_override element defines a group of snap-in attributes that has the same value.

Following is an example of attribute_overrides declared in properties.xml:

```
<service>
    <smgr>
```

```
        <attribute_overrides>
        <attribute_override>
                <factory_override name=”xxxx”>
                        <displayName>phoneNumber</displayName>
                        <type>STRING</type>
                        <validation name="type">
                                <type>STRING</type>
                        </validation>
                        <admin_visible>false</admin_visible>
                        <admin_changeable>false</admin_changeable>
                        <factory>
                                <value>2222</value>
                                 <user_changeable>false</user_changeable>
                        </factory>
                </factory_override>
                <override_snapin_attribute>
                        <snapin_name>MultiChanBroadcastService</snapin_name>
                        <attribute_name>smsFrom</attribute_name>
                </override_snapin_attribute>
        </attribute_override>
        </attribute_overrides>
    </smgr>
</service>
```

Definition of attribute_overrides includes the followings parts:

- attribute_override
- factory_override
- override_snapin_attribute

Definition of factory_override includes the following parts:

- Attribute name: The name of the attribute.
- Display name: The name of the attribute in the Service Profile editor.
- Help Info: The descriptive text of the attribute in the Service Profile editor.
- Type: The type defined in the following Avaya Breeze® platform attribute types section.
- Admin visible: The option to display the attribute in System Manager. This value must be “false”.
- Factory: This section describes the default values of the attribute. It includes the “value” and “user_changeable” parts. “user_changeable” must be “false”.

Definition of override_snapin_attribute includes:

- snapin_name: The snap-in loaded on System Manager.
- attribute_name: The name of the attribute to override.

Notes:

- The attribute_override element must include at least one factory_override element.
- The sub-element of factory_override must be one of the following: STRING, INTEGR, or BOOLEAN
- The attribute_override element must include at least one override_snapin_attribute element.

- For each attribute_override element, the type for the attribute specified by the attribute_name element for the snap-in specified in each override_snapin_attribute element must be the same as the type specified in the factory_override element.
- If the specified snap-in is currently loaded on System Manager, the factory default value of the attribute specified by the attribute_name element is updated with the value provided in the factory_override element.
- For the Password attribute, use ENCRYPTED_STRING as validation type.

How to read service profile attribute values

First, get an instance of ServiceData from the CollaborationDataFactory. Then, use ServiceData's getServiceAttribute method to retrieve the value defined for the attribute in the template. The getServiceAttribute method takes two parameters:

- attributeName: the name of the attribute for which to retrieve the value.
- userAddress: the user for which you want to retrieve the attribute value in the format handle@domain. It may be desirable to use the result of the Participant.getAddress() method as input for this parameter.

If there is no value for the user's service profile, the value for the attribute administered for the cluster will be returned. If no there is no value administered for the cluster, then the global value administered is returned. If no global value is administered, then the default value is returned.

Sample code

For example, the following code snippet modifies the call listener from the first example to retrieve and log the attribute value declared above.

```
@TheCallListener

public class CallListener extends CallListenerAbstract {

    private final Logger logger = Logger.getLogger(CallListener.class);

    @Override
    public final void callIntercepted(final Call call) {
            ServiceDescriptor svc = ServiceUtil.getServiceDescriptor();
            ServiceData svcData =
CollaborationDataFactory.getServiceData(svc.getName(),svc.getVersion());
            String attrValue =
svcData.getServiceAttribute(call.getCallingParty().getAddress(), "displayString");
            logger.info("attribute value was " + attrValue);
    }
}
```

Service cluster/service global attribute values

If you do not want to get attribute values based on users, use the same method, getAttribute() but specify just the attribute name. It works the same way except the service profiles are not checked.

You might want to protect certain data, such as passwords. An attribute can be defined as an encrypted attribute. Such an attribute is stored internally in encrypted form, and the value is masked on the Attribute Configuration form:

```
<attribute name="accountPassword">
        <displayName>Account Password</displayName>
        <helpInfo>The password for this account.</helpInfo>
        <validation name="EncryptedString">
                <type>ENCRYPTED_STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <admin_changeable>true</admin_changeable>
        <factory>
                <value></value>
                <user_changeable>true</user_changeable>
        </factory>
</attribute>
```

The method getServiceEncryptedAttribute would be used to retrieve an attribute that has been defined as encrypted.

Sample code

As an example, the following code snippet retrieves a username (clear text) and password (encrypted):

```
final class MyAttributeReader {
      public String getUsername() {
                ServiceData svcData = CollaborationDataFactory.getServiceData(“FooService”,
            “2.0.0.0.0”);
                return svcData.getGlobalServiceAttribute(“username”);
      }
      /**
      * Return the decrypted value for some data that was stored in encrypted form.
      */
      public String getPassword() {
                ServiceData svcData = CollaborationDataFactory.getServiceData(“FooService”,
            “2.0.0.0.0”);
                return svcData.getGlobalServiceEncryptedAttribute(“accountPassword”);
        }
}
```

Snap-in URL

The administrator can select a Snap-in URL from the Service URL column on the Cluster Administration page. This will navigate to one of the landing pages of the snap-ins installed on the cluster. Developers can specify one or more home or landing pages for their snap-in by adding them using the cutthrough_url property in the properties.xml as shown in the example.

Example

If the home page or landing page of the snap-in is “admin.html” or “index.html”, then the below sample code needs to be added in the properties.xml after the <smgr> tag.

```
[service xmlns=http://archiveschemas.aus.avaya.com/properties
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CES2" version="3.1.0.0.0"
        application="CES2-3.1.0.0.0"
        xsi:schemaLocation="http://archiveschemas.aus.avaya.com/properties properties.xsd"]
        [smgr]
        .......
        .......
        .......
        [/smgr]
        [cutthrough_url]
                [url_display_name]Admin URL[/url_display_name]
                [url]admin.html[/ur]
        [cutthrough_url]
        [cutthrough_url]
                [url_display_name]Designer URL[/url_display_name ]
                [url]index.html[/ur]
        [cutthrough_url]
[/service]
```

Cluster attribute values

Cluster Attributes have a name that is similar to “Service Cluster Attributes” but they are really quite different. Service Cluster Attributes are values that are service-specific but are provisioned at the cluster scope. Cluster attributes, on the other hand, are values that are defined by the administrator for the Avaya Breeze® platform cluster as a whole. They are not specific to any particular service and are not defined by the developer of the snap-in. Cluster attributes are accessed through the com.avaya.collaboration.businessdata.api.ClusterData class.

Attribute notifications

Avaya Breeze® platform DAO provides a means for snap-ins to request notification on changes to its attributes.

Usage

Snap-in needs to define listeners to listen to attribute change notifications. The listener can be defined by extending “com.avaya.zephyr.platform.dm.AbstractDMListener” class. This class has a callback method called “#objectChanged”, which gets invoked whenever there is a change in snap-in's attribute value. Below is an example of attribute change listener:

```
public class TestDaoListener extends AbstractDMListener
{
      private static TestDaoListener listener = new TestDaoListener();
      private TestDaoListener() {
}

public static TestDaoListener getInstance()

{

          return listener;

}

@Override
public void objectChanged(Object oldObject, Object newObject)
{
       if (oldObject instanceof DefaultAttribute || newObject instanceof DefaultAttribute)
       {
       //Objects are of type DefaultAttribute, when the attribute value changed at "Service Global"
       level
       }
       if (oldObject instanceof ClusterDefaultAttribute || newObject instanceof
       ClusterDefaultAttribute)
       {
       //Objects are of type ClusterDefaultAttribute, when the attribute value changed at "Service
       Clusters" level
       }
       if (oldObject instanceof AusAttribute || newObject instanceof AusAttribute)
       {
       //Objects are of type AusAttribute, when the attribute value changed at "Service Profiles"
       level       }
       }
}
```

Once you get the notification, use the apis serviceData.getServiceAttribute(attributeName) or serviceData.getServiceAttribute(useraddress, attributeName) to get the latest attribute values as shown below.

```
@Override
```

```
        public void objectChanged(Object oldObject, Object newObject) {
        if ((newObject instanceof DefaultAttribute) || (oldObject instanceof DefaultAttribute))
        {
        if (newObject != null)
        {
        if (((DefaultAttribute) newObject).getAttributeName().equalsIgnoreCase("attribute1"))
        {
        LOGGER.info("New Value:" + svcData.getServiceAttribute("attribute1"));
        }
        }
        else
        {
        if (((DefaultAttribute) oldObject).getAttributeName().equalsIgnoreCase("attribute1"))
        {
        LOGGER.info("New Value:" + svcData.getServiceAttribute("attribute1"));
}
}
}
}
```

If you are changing the attribute values only at the cluster and global level, then the api serviceData.getServiceAttribute(attributeName) can be used. If you are also interested in the attribute values at the Service Profile level, then you should use the api serviceData.getServiceAttribute(attributeName). Refer to the Javadoc for more information on how these apis work.

The listeners need to be registered with DAOs (AusServiceDAO / AusAttributeDAO) to get notifications.

For notifications of attribute value changes at “Service Global” and “Service Clusters” level, the listener needs to be registered with “AusServiceDAO” and to listen to attribute change notifications at ““Service Profiles”” level, with “AusAttributeDAO”. And the listeners should also be removed during snap-in uninstallation.

The registration and removal of listeners can be done at “#init” and “#destroy” methods of “ServiceLifeCycle” respectively.

Register an object with *DMFactory* before adding any references to the DAO classes. Registering objects can be done using the #init method of the ServiceLifeCycle or #postConstruct method of a startup bean. Register an object that remains throughout the snap-in lifecycle. This ensures that the DAO objects being referenced in the service will not get garbage collected as long as the 'object' exists.

Failing to do this registration will result in warning messages in platform logs, such as "**WARN - Calling component not registered with the DMFactory: ClassLoader Name....**".

Below is an example of registration and removal of listeners with DAOs:

```
@TheServiceLifeCycle

public class MyServiceLifeCycle implements ServiceLifeCycle

{

        @Override public void init()

        {

                // Registering the class to DMFactory
                DMFactory.getInstance().register(this);

                // Registration of Listener with "AusServiceDAO" for notifications of attribute value
        changes

                // at "Service Global" and "Service Clusters" level

DMFactory.getInstance().getDataMgr(AusServiceDAO.class).registerListener(TestDaoListener.g
etInstance());

                // Registration of Listener with "AusAttributeDAO" for notifications of attribute
        value changes
                // at "Service Profiles" level
DMFactory.getInstance().getDataMgr(AusAttributeDAO.class).registerListener(TestDaoListener
.getInstance());

        }

        @Override public void destroy()

        {

        // Removal of Listener from "AusServiceDAO"

DMFactory.getInstance().getDataMgr(AusServiceDAO.class).removeListener(TestDaoListener.get
Instance());


        // Removal of Listener from "AusAttributeDAO"

DMFactory.getInstance().getDataMgr(AusAttributeDAO.class).removeListener(TestDaoListener.g
etInstance());
        }
}
```

Logger

The log files for a particular service can be viewed by typing ce dlogv<*servicename*> (e.g., ce dlogv myCEService) from the command line on the Avaya Breeze® platform server.

Debug logging (Fine level and lower) can be enabled by typing ce dlogon<*servicename*> and disabled by typing ce dlogoff <*servicename*>.

The logger class provided for collaboration services is com.avaya.collaboration.util.Logger. This class logs messages to the log file for your service. To obtain a logger for a class in a service, call Logger.getLogger(YourClass.class).

To log a message, call the method of the Logger object named after the logging level that you want to log at. The logger provides 7 logging levels which are as follows.

- Fatal: Non-recoverable error
- Error: Recoverable error.
- Warn: Not an error, but might indicate something is not as it should be
- Info: Logging that an event occurred in your service
- Fine: coarsest debugging info
- Finer: finer debugging info
- Finest: finest debugging info

Note that ""Info"" is the lowest level that is enabled by default. Also note that excessive logging can impact performance. In general, you should log only significant events at the "Info" level, and use the "Fine", "Finer", and "Finest" for more detail. The log level can be changed on the fly to be more verbose; it can also be changed on the fly to return to the default level. In a development environment, you might choose to leave logging at the most verbose level. In a production environment, you might change the log level to verbose, run a test to collect data, and then change back to the default log level.

The logger also provides methods that check if the fine, finer, or finest logging level is enabled (e.g., isFinerEnabled()). This can be used to save the time that would be spent constructing a log message that will not be used.

Sample code

For example, this statement from the attribute example, logger.info("attribute value was" + attrValue); will replace a line like the following in the System Manager log:

2013-02-08 13:48:00,070 [SipContainerPool : 1] com.mycompany.testService.CallListener INFO – testService-2.0.0.0.0 – attribute value was [attribute value]

Each service is allocated 100 MB of logging space. To request a different amount of space, you may do so in the properties.xml file. Locate the code phrase <log_space>10MB<log_space>. Uncomment it and put in the requested allocation in megabytes. So, to request 200 megabytes, the line would look as follows: <log_space>200MB<log_space>. The log for your service will be located in /var/log/Avaya/services/ServiceName/. This will create 20 files of 10 MB each for this service.

Raising alarms

Snap-in developers are able to raise alarms that are specific to their snap-in. These alarms will be sent to System Manager and/or other Network Management Systems. Alarms are defined in the <*servicename*>-svar/src/main/resources/alarms.xml file for a given service.

Event names are generated in the <*snap-in name*>_<*event code*> format. In a svar, the maximum characters of event code that can be defined is 12. If an alarm is provisioned for clearing then “CLR_” is added to the event code, which increases the character count by 4, which makes the length of the event code 16. In this case, the event name is generated in the format: <*snap-in name*>_<CLR_*event code*>. For example, in case of the Whitelist sample service, Whitelist_DB_ERROR_01 can be a sample event name.

Important:

The maximum length supported for snap-in event names is 32 bytes.

The Clear flag makes the alarm clearable by System Manager by automatically creating an entry of <CLR_*event name*> for <*event name*> alarm.

The following is an example of an alarm definition from the Whitelist sample service:

```
<xml version="1.0" encoding="UTF-8">
<alarms xmlns:p=http://archiveschemas.aus.avaya.com/snapinalarms
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://archiveschemas.aus.avaya.com/snapinalarms alarms.xsd ">
    <alarm>
        <eventcode>DB_ERROR_01<eventcode>
                <alarmDisplayText>Whitelist service couldn't connect to the
        DB.<alarmDisplayText>
                <severity>major<severity>
                <clearFlag>false<clearFlag>
        <alarm>
<alarms>
```

The sample service further shows that the alarm can then be raised with a simple log statement:

```
catch (final Exception e)
    {
            entityManager = null;
            logger.error("initializeEntityManager exception=", e);
            logger.logEvent("DB_ERROR_01");
    }
```

When developing alarm-related snap-ins changing the throttling value is important, as the same alarm is not raised within 12 hours, which is the default throttle period. To help debug alarms getting raised, either change the throttle value to lesser time or disable throttling. See “Changing the throttle value” or “Disabling throttling”. You can also configure the throttle time for a specific event. See “Configuring the throttle time for a specific event”.

Note:

Contact Avaya Services to change the alarm throttle value and to disable alarm throttling.

Throttling changes are lost after service removal as it is specific to a service. However, the system-level attribute changes in AlarmThrottle.properties get reset after an upgrade.

Avaya Breeze® platform does not restrict the number of alarms in each snap-in, but it is better to have a maximum of 100 alarms for each snap-in.

Guidelines for raising alarms

Alarms.xml must adhere to the following three conditions so that the snap-in loads without a failure:

The SNMP Notification OID is the unique identifier for a trap/alarm. Alarms usually have two notifications OIDs - one for setting the alarm and one for resolving the alarm. Setting the OID is typically not required. The only reason to set the OID would be if:

- You are not an Avaya developer and need to represent your snap-in to the SNMP manager under your enterprise / product IDs.
- You are an Avaya developer and need to represent your snap-in to the SNMP manager under your product ID.

If you do not define the Notification OID, your alarm will automatically be represented under the Avaya enterprise and the Avaya Breeze® platform (CE) product ID. If you do define a Notification OID, ensure that it is unique for your enterprise/product.

If you wish to represent a different organization than Avaya with your alarm, you must set the OrgType (p:organization) tag. If you set this tag, it is mandatory to specify a notification OID as well. Below is an example of how to do both of those.

For more information on Notification OIDs, please see Wikipedia: https://en.wikipedia.org/wiki/ Object_identifier_.

```
<p:organization>
<p:enterpriseToProductId>123456.2.83</p:enterpriseToProductId>

<p:enterpriseToProductIdentifier>mycompany.products.samplep:enterpriseToProductIdentifier<
/p:enterpriseToProductIdentifier>
</p:organization>
    <p:alarm>
            <p:eventcode>EVENT_001</p:eventcode>
            <p:alarmDisplayText>{1}:{2}p:alarmDisplayText>{1}:{2}
            </p:alarmDisplayText>
            <p:severity>major</p:severity>
            <p:notificationOid>.1.3.6.1.4.1.123456.2.83.0.1</p:notificationOid>
            <p:clearFlag>true</p:clearFlag>
<p:clearNotificationOid>.1.3.6.1.4.1.123456.2.83.0.101</p:clearNotificationOid>
    </p:alarm>
```

In this example, 123456 is the enterprise number assigned to "mycompany" by IANA. 5 corresponds to "products" and 20 corresponds to "sample".

Alarm severity can be defined using the <p:severity>...</p:severity> tag. Severities are: Minor, Major, Warning, Critical, and Normal.

Note:

Contact Avaya Services for configuring of throttle time for a specific event.

Avaya Breeze® platform application programming interface

Up to this point, we have covered the basics of the Avaya Breeze® API, but there is much more. Here is a short introduction to the other parts. Details to each part including samples of how to use it are contained in the Javadoc documentation. The sample code is included as its own package named after the API with a .sample appended. Explore and use any or all parts you need to quickly and easily write your Avaya Breeze® Services.

Collaboration Call API

This API allows you to process incoming calls and launch outgoing calls. We've already shown you a little, but if you want more, see the Javadoc under package com.avaya.collaboration.call.

One concept you will see in the Call API is the concept of a "UCID" (Universal Call ID). This is similar in nature to the call ID that can be retrieved through Call.getId(). The difference is that the Universal Call ID is also accessible to other entities in Avaya Aura®. For instance, Application Enablement Services applications and Avaya Aura® Experience Portal both have access to the UCID. UCID is also used extensively in the Avaya Aura® reporting platforms. The Call ID, however, is scoped only to a Avaya Breeze® cluster.

Collaboration Bus API

The collaboration bus API is a simple way to send and receive messages between services. If you are interested, please see the Javadoc located under the package com.avaya.collaboration.bus. The com.avaya.collaboration.bus.sample contains a sample client and connector.

Eventing API

The eventing API allows services to produce and/or subscribe to events in a loosely coupled fashion. The event types and semantics are not defined by the eventing API itself. The API is quite happy to accept any event family, type and message bodies passed by the producers and consumers.

One might wonder why this API is needed in addition to the Pub/Sub channels provided by the Collaboration Bus API. There are a few key differences between these two APIs.

- Subscription establishment/duration:
  - Pub/Sub: subscriptions are defined in the properties.xml file and are therefore static for the entire lifecycle of a service.
  - Eventing: subscriptions can be established / cleared dynamically at any point during the service lifecycle.
- Filters:
  - Pub/Sub: there is no support for fine-grained filters. A service is invoked for every event sent on the channel.
  - Eventing: a service can specify fine-grained filters based on specific users, calls or other criteria specific to event families.
- Locality of event publishers:
  - Pub/Sub: the publisher of events must be executing as a CE service. If any external events are to be published, a snap-in must expose its own web service interface to receive those events.
  - Eventing: An Eventing Connector is provided. This Connector has a pre-defined REST interface that allows remote applications to publish events

directly into the Eventing Framework. The Connector also has a REST interface to allow remote applications to subscribe for events that are then delivered via HTTP POST.

- Consumer Private Data:
  - Pub/Sub: there is only a single subscription per service so there is no easy way to have service-specific data associated with an event.
  - Eventing: a service can provide “Consumer Private Data” with a subscription. Services will often have some data specific to that service that will be related to events for specific subscriptions (for example., some data associated with a user subscription). This feature enables such data to be provided to the service without the service having to maintain a separate map.

One of the primary concepts in the Eventing API is that of Event Families and Event Types. An example of an Event Family is the Call Event family. Examples of Event Types are Call Alerting, Call Answered, and Call Ended. It is important to note that Call Events, while prepopulated and generated by an Avaya provided service (Call Event Control) are no more integrated into the Eventing Framework than would be an Event Family produced by a third party. The Eventing API has no semantic knowledge of Call Events. It simply dutifully relays the Call Events from the Call Event Control Service to any services that have subscribed for such events.

Another important concept in the Eventing API is the idea that a subscription is scoped to a cluster and not to a specific server within a cluster. If a service subscribes to an event on server A within a cluster, and that event occurs on server B, the subscribing service will be notified of that event. It is important to note, however, that the instance of the service running on server B would be notified of this event rather than the instance that actually subscribed. You should bear this in mind when designing your logic. Anything stored in local memory on the subscribing server may not be accessible when actually processing the event. However, Consumer Private Data does cross nodes. Any private data provided when subscribing on server A in this example would be provided to the service on server B when an event occurs for this subscription.

The Eventing Framework has a mechanism to treat subscriptions as being duplicates of each other and ensures that only one event will be sent to a subscriber. This is to handle the case where a service subscribes to events on startup and will therefore subscribe on each server in a cluster. In such cases, it is desirable to treat those multiple subscriptions as a single subscription so that only a single service instance is notified when an event of interest occurs. There are very specific criteria to determine if a subscription is a duplicate. Please check the Javadoc for the com.avaya.collaboration.eventing package for a detailed description of what qualifies as a duplicate subscription.
There are 2 primary roles with respect to Eventing: producers and consumers. Event Producers needn’t know anything about who (if anybody) is subscribed for events. They simply use the Producer API to publish their events with the proper event family, type and metadata. Similarly, Event Consumers subscribe for events without the need to know which service (or services) is the producer of those events.

Given the earlier discussion of Call Events, you might be wondering why one would use this method of receiving call events rather than functioning as a Call Intercept service. There are several reasons why you might choose to subscribe for call events rather than intercept calls:

- Reduced call latency. Call Intercept services are invoked serially, and an intercepted call is not allowed to proceed to the called party until each and every Call Intercept service has executed its logic. However, some services don’t need to actually perform their logic

before the call is sent to the called party. If such services subscribe for Call Events, the call will be allowed to proceed before the service logic is invoked, thus reducing latency.

- Textual events. Some services will want to store call events in a database or send them to a remote system. If using the Call Intercept interface, these sorts of services would have to define their own textual format for the events in order to send them across the wire. This is not an issue with Call Events, as they are sent in a JSON format and a JSON schema is available for those events. If you'd rather work with Java objects than JSON strings, never fear! Tools such as Gson exist that allow you to easily deserialize JSON strings into Java objects.

Keep the following in mind when using Call events.

- Depending on configuration, subscribers may be notified that a call has been answered when in actuality it has been intercepted by a snap-in or workflow that then played an announcement. No subsequent answered event would be received if the call is sent along to an endpoint.
- If a snap-in/workflow is performing Flexible Call Leg Control operations, a subscriber to call events may think a call has completely dropped when only one party has dropped, or it may not be aware of a change of participants.

Examples of how to produce and consume events can be found in the Javadoc in the com.avaya.collaboration.eventing.sample package. An example of an HTTP event producer can be found in that same package.

Eventing Frameworks

Avaya Breeze features 2 types of Eventing Frameworks.

Eventing Framework (EF)

EF is the original event delivery method provided through IBM Service Integration Bus (SIB) that runs in the context of the same JVM (WebSphere) as most Avaya Breeze snap-ins. The event delivery method increase the speed of the publish and consume events within a single Avaya Breeze node.

Reliable Eventing Framework (REF)

REF uses a central, highly available message broker group to distribute events across the Avaya Breeze nodes and clusters. REF guarantees the event delivery reliability through event persistency and replication and load balancing of events across multiple consumers on the same event subscription.

For more information about consuming events from REF, see Breeze ReliableEventStreaming Adapter User Guide.

Collaboration Media API

The Media API allows you to add the ability to play announcements and collect digits as part of your call processing. If the Real-Time Speech Snap-in has been installed, you can also perform text to speech (TTS). These operations can be carried out at any point during a call: when the call is intercepted, during alerting and after answer.

Note:

For best performance, the recommended media file format of a recorded announcement is a 16-bit, 8 kHz, single channel (mono), PCM WAV file. For more information about other supported formats see *Implementing and Administering Avaya Aura® Media Server*.

Now let's look at some code snippets that illustrate the use of the media operations. Please also see the Javadoc located under the package com.avaya.collaboration.call.media as well as a sample service located at com.avaya.collaboration.call.media.sample.

Please see the Javadoc located under the package com.avaya.collaboration.call.media as well as a sample service located at com.avaya.collaboration.call.media.sample.

Play Announcement:

There are a few different ways to play announcements. They are distinguished by the format of the "source" parameter on the PlayItem object:

1. The recorded announcement can be accessible via HTTP, either as part of a snap-in or on a separate HTTP server. In this case, the source URI would simply look like an HTTP URL: http://www.mycompany.com/announcements/greetings/welcome.wav.
2. The recorded announcement can be populated in the Avaya Media Server content store. In this case, the source URI would have the following format where "ns" stands for namespace and "cg" stands for content group. An administrator must have previously populated the wave file on AAMS: cstore://welcome?ns=announcements&cg=greetings
3. If the Real-Time Speech Snap-in has been installed and if a Speech Server like Nuance has been configured with your Avaya Media Server, you can use Text to Speech (TTS). This can be done by simply putting the text string into the source URI: "Welcome to my company."

The following snippet shows how to construct a PlayItem using the content store format and play it to the calling party:

```
final PlayItem playItem = MediaFactory.createPlayItem();
    final MediaService mediaService = MediaFactory.createMediaService();
    playItem.setSource("cstore://welcome?ns=announcements&cg=greetings");
    playItem.setInterruptible(true);
    playItem.setIterateCount(1);
    playItem.setDuration(5000);
    final UUID requestID = mediaService.play(participant, playItem,
            myMediaListener);
```

Collect digits:

There are 2 ways that you can collect DTMF digits from a caller or called party in a call. You can collect digits without specifying any announcement at the same time. Alternatively, you can invoke the promptAndCollect method that will play an announcement then collect digits immediately after. The single promptAndCollect operation is the more common way of doing things and is what is shown below.

The PlayItem that is passed would be constructed in exactly the same fashion as is used for the play method. A DigitOptions object is also required to invoke promptAndCollect. The first few parameters in the DigitOptions class are about when to stop the collection and return results. These values are all optional, and the first to be satisfied will cause the collection to be terminated.

The NumberOfDigits value defaults to 1. If you want to keep collecting digits until the termination key is pressed or until there is a timeout, you would have to set this to a very high number. In most cases, however, you'll want to set an upper bound anyway so this will not be an issue.

The TerminationKey is quite simple to understand. When this key is pressed by the user, the collection completes, and results are returned. By default, there is no termination key.

A timer is started with value Timeout (in milliseconds) immediately after the collectDigits or promptAndCollect operation. If this timer expires before a digit is entered, the collection terminates. The default value is 60000 milliseconds (one minute).

The FlushBuffer parameter is an indication of whether any digits collected prior to the invocation of collectDigits or promptAndCollect should be discarded. If set to true, all digits in the buffer will be discarded. If false, they will be retained.

```
final DigitOptions digitOptions = MediaFactory.createDigitOptions();
    digitOptions.setNumberOfDigits(MAX_DIGITS);
    digitOptions.setTerminationKey("#");
    digitOptions.setTimeout(MILLIS_TO_WAIT);
    digitOptions.setFlushBuffer(true);
    final UUID requestID = mediaService.promptAndCollect(participant,
            playItem, digitOptions, myMediaListener);
```

The collect digits operation can be used in a 2 party call to, for instance, collect a credit card from one of the parties. If an announcement is played towards the other party, the second party would not be able to hear what was being dialed by the first party.

Stopping a media operation:

Sometimes, it will be desirable to stop a media operation that is in progress. For instance, if the Work Assignment Element indicates that an agent is available to take a call, you would want to stop music from playing to the caller before routing that caller to the agent. To do this, you would utilize the unique request identifier (UUID) that was returned to you when invoking the play method. You might have saved this identifier as an attribute on the Call object.

Controlling a call on completion of a media operation:

In many cases, you will want to take some action after a media operation completes. For instance, you may want to play an announcement to a caller asking if a call is urgent, then take varying actions (allow, divert, terminate) based on input from the user. If you want to take action on a call, you first must have saved a handle to the Call (or the string call ID) someplace. The easiest way to do this is to have a field in your MediaListener implementation class that contains a handle to the call. The following code snippet illustrates that concept:

```
final MediaListener myMediaListener = new MyMediaListner(call);
    ...
    final UUID requestID = mediaService.promptAndCollect(participant, playItem, digitOptions,
            myMediaListener);
```

Your implementation of the digitsCollected might then look like:

```
void digitsCollected(UUID requestId, String digits,
    DigitCollectorOperationCause cause)
{
```

```
    if(digits.equals(URGENT))
    {
            call.allow();
    }
    else if(digits.equals(NOT_URGENT))
    {
            call.terminate();
    }
}
```

Important limitation: no cross-server media invocations:

A very nice aspect of the Call Manipulation API is that call control operations can be invoked across servers in a cluster. That means that if a call is handled by server A in a cluster, then an HTTP message arrives at server B in that same cluster, the service logic in server B can control the call on server A. This is all handled automatically by the API. Just get a Call object from the CallFactory on server B and operate on it as if that call was local. However, that same functionality is not available for media and speech operations. If there's a chance that your service might receive HTTP events on one server that will need to operate on a call on another server, you'll need to redirect that HTTP event on your own. A utility has been provided that will make this easier for you:

```
if (!CallProperties.isHostedOnLocalNode(myCall)
{ String remoteNodeIp = CallProperties.getNodeThatHostsThisCall(myCall)
 /* Redirect HTTP request to the correct node. */ }
```

Redirecting an HTTP request to the correct node can be done either by redirecting to an FQDN that corresponds to the IP address of the remote node, or by redirecting to the cluster FQDN using an affinity query parameter that contains the IP address of the server. For example:

https://myBreezeCluster.example.com/services/myBreezeService/resource?affinity=xxx.x.x.xx.

Note:

It is not recommended to redirect to a URL based on an IP address, since TLS certificates are generally not issued for IP addresses, common names, or subject alternative names.

Collaboration Conference API

Provides classes and interfaces for scheduling conferences, including ones that can begin right away. It also includes a raw interface to Scopia. You must install the Scopia Connector that is pre-loaded on the system to use the Conference API. Please see the Javadoc located under the package com.avaya.collaboration.conference.scheduled.

Collaboration Email API

This API should be used to send Emails. You must install the Email Connector service that is pre-loaded on the system to use the Email API. The Javadoc for this API is located under the package com.avaya.collaboration.email. There is a useful sample there also.

Collaboration SMS API

This API should be used to send SMS messages. You must install the SMS Connector service that is pre-loaded on the system to use the SMS API . The Javadoc for this API is located under the package com.avaya.collaboration.sms. There is a useful sample there also.

Collaboration SIP Header Manipulation API

Provides classes and interfaces for doing SIP header manipulation. Please see the Javadoc located under the package com.avaya.collaboration.call.sip.

Collaboration System Status API

The System Status API provides methods for checking CPU utilization, overload status and other system information from a service. Note that the overload status should generally be checked before starting new work. If the system is overloaded, adding more work will likely further degrade performance. The Javadoc is located under com.avaya.collaboration.util.

Collaboration Service Data API

The Service Data API allows services to access global and service attributes. The Javadoc is located under the package com.avaya.collaboratation.businessdata.api.

Collaboration User Data API

The User Data API allows services to access data associated with their phone numbers and handles. The Javadoc is located under the package com.avaya.collaboratation.data.api.

Collaboration Logging API

The Logging API allows services to log data based on service name and version . The Javadoc is located under the package com.avaya.collaboratation.util.logger.

Collaboration Service API

The Logging API allows a service to obtain data about itself, like name, version and the version of the SDK used to build the service. The Javadoc is located under the package com.avaya.zephyr.platform.dal.api.

Node Status API

Avaya Breeze® adds the capability for a snap-in to request the status of another Avaya Breeze® node in its cluster. The status indicates the ability of a node to process calls. When the status is UP, the node is able to process calls; when the status is DOWN, the node is not able to process calls.

Use the following API to retrieve the status of all nodes in a cluster:

```
package com.avaya.collaboration.cluster;
List<NodeStatus> Cluster.getNodeStatus();
```

Session Manager (SM) uses OPTIONS requests to both monitor individual Avaya Breeze® nodes and to inform individual Avaya Breeze® nodes about the status of all Avaya Breeze® nodes in a cluster. By default, the OPTIONS requests are sent once every 900 seconds (15 minutes). This means that Avaya Breeze® node and a snap-in using the Node Status API might have the incorrect status of another Avaya Breeze® node for up to 15 minutes.

When a snap-in is using the Node Status API, Avaya recommends that you increase the frequency of the OPTIONS requests that SM sends to the Avaya Breeze® nodes in the cluster from once every 900 seconds to every 10 seconds so that a status change is detected faster.

Administrators can change how frequently SMs send OPTIONS requests. For information on how to change the rate of the OPTIONS requests on SM for this feature, see *Administering Avaya Breeze® platform*.

Send Digits API

Using sendDigits method in MediaService, you can send digits (DTMF tones) to a participant in a call. The method also allows you to register a MediaListener instance, so that you get a callback to perform some actions after the send digits operation completes. The sendDigits API supports sending the set of digits { A,B,C,D, 0-9, *, # }. A typical use of this API will look like the below code snippet.

```
//Obtain a participant
Participant participant = ... ;
//The digits to be sentString digits = "D19175";
//create an instance of MediaListenerMediaListener
listener = new MediaListenerAbstract()
{
    public void sendDigitsCompleted(final UUID requestId, final SendDigitsOperationCause cause)
    {
            //it's invoked when send digits operation gets completed
    }}
    ...
//obtain an instance of MediaServiceMediaService
mediaService = MediaFactory.createMediaService();
//Perform send digits operation
mediaService.sendDigits(participant, digits,listener);

...
```

Speech API

The speech search portion of the Speech API is operational only if you have installed and licensed the Real-Time Speech snap-in. This powerful API enables you to perform speech queries on live one or two party conversations so that you can be notified when somebody speaks a phrase of interest. If you also have installed and licensed a speech server such as Nuance, you will additionally be able to use Automatic Speech Recognition by using the VoiceXMLDialog methods. Use of the VoiceXMLDialog portion of the API does not require the Real-Time Speech snap-in to be installed.

The Speech Search operations are sufficiently detailed such that they are not described in this guide. Instead, they are described in the separate Real-Time Speech SDK. Similarly, no description will be provided on how to construct a VoiceXML script. The following snippet illustrates the use of the VoiceXMLDialog methods, however.

```
final SpeechService speechService =
            SpeechFactory.createSpeechService();
    final VoiceXMLDialogItem = SpeechFactory
            .createVoiceXMLDialogItem();
    final URI scriptURI = new URI(voiceXMLScript);
    voiceXMLDialogItem.setVoiceXMLScript(scriptURI);
    speechService.startVoiceXMLDialog(participant, voiceXMLDialogItem,
            CMATestListenerFactory.createVoiceXMLDialogListener(call));
```

SSLUtil API

The SSLUtil API allows to create SSLContext for TLS connections using Avaya Breeze® platform's trust store and key store. See the Javadoc located in the

com.avaya.collaboration.ssl.util.SSLUtilityFactory package. There are also some useful samples in the com.avaya.collaboration.ssl.util.sample package.

If there is no TLS version specified explicitly then by default a TLS version will be used as mentioned in the section “How to decide which TLS version to be used”.

How to get the original HTTP request IP and scheme

A snap-in that consumes HTTP requests in some cases could be interested in the Original HTTP Request IP:Port and Scheme that was used to make the HTTP request to the Avaya Breeze® snap-in. This could be specifically useful when an HTTP request was made through multiple HTTP reverse proxies in a network and a load balancer.

An Avaya Breeze® snap-in can get this information by reading the “Host” and “Scheme” headers.

Example scenario: A snap-in is accessed from a browser using the URL : http://BreezeCluster-IP/services/<snap-in name>/index.jsp and the request gets to Avaya Breeze® platform, which is on Breeze-IP.

In the above scenario if a snap-in wants to know the Breeze-Cluster-IP, it can get that from the “Host” header in the HTTP request and get the Scheme as http or https from the “Scheme” header in HTTP request.

How to get the HTTP/HTTPS proxy settings

Avaya Breeze® platform provides an API *HttpProperties* class for snap-in developers to retrieve the HTTP/ HTTPs proxy setting information on the platform. When a snap-in is designed to send an HTTP/ HTTPs request out of the customer’s network boundary to the internet, the snap-in should use the API to get proxy settings. All requests must pass through the forward (outbound) proxy, if any, which will send the requests to the destination, and forward the received responses back to the snap-in.

HTTP header X-Frame-Options

Snap-ins can set the HTTP header “X-Frame-Options" in the response as per their usage.

If not added, Avaya Breeze® platform uses SAMEORIGIN as the value. Other possible values for this response header are “DENY”, "ALLOW-FROM uri” where uri is any HTTPS URI. For example, https://example.com/.

HTTP headers in responses

When sending HTTP responses, Avaya Breeze® platform adds the following headers with the corresponding values as mentioned in the below table:

Header	Default Header Value	Additional comments and Possible values which snap-ins may want to specify
X-Frame-Options	SAMEORIGIN	"DENY" - To prohibit any domain from framing the content. "ALLOW-FROM *uri*" - Permit a specific *uri* to frame the content.
Strict-Transport-Security	"max-age=31536000; includeSubdomains"	Snap-ins might want to change max-age value in seconds or remove includeSubdomains tag.
X-Content-Type-Options	"nosniff"	
X-XSS-Protection	"1; mode=block"	"0" disables XSS filtering, this should be avoided.
Cache-Control	"no-cache,no-store"	This header specifies directives for caching the response. There are various different values which snap-ins may want to specify for this header. Please see all the valid values at https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control
Pragma	"no-cache"	Pragma is HTTP/1.0 is an implementation-specific header - Should not be overridden, see Cache-Control instead.
Expires	"0"	HTTP-date timestamp can be used instead of "0" E.g. Expires: Mon, 27 Nov 2017 02:44:00 GMT

If a snap-in wants to change the value for any of the above headers in its HTTP response, it can do so and Avaya Breeze® platform will retain the value set by the snap-in.

For all of the above headers, Avaya Breeze® platform advises to use default values and override them only when it is absolutely necessary for snap-in functionality.

When to use the session affinity parameter for HTTP load balancing

By default, the Avaya Breeze® platform HTTP load balancer uses a round-robin algorithm to distribute incoming HTTP requests among Avaya Breeze® nodes in a cluster. Snap-ins that are stateless, including those that store state in an external database or memory grid, are fine using the default round-robin algorithm. However, stateful snap-ins require affinity of HTTP requests to a particular Avaya Breeze® platform server. In these cases, you must enable the session affinity attribute on the Avaya Breeze® cluster. The session affinity attribute is present in the Cluster Attributes section of the Cluster Editor page in System Manager.

When you enable session affinity, HTTP requests are targeted to a particular Avaya Breeze® platform server in two ways:

- By default, a hashing algorithm is used on the source IP address such that all requests from a given IP address are routed to the same Avaya Breeze node. This works well in cases where the senders of the HTTP messages are end-user devices and the real IP address of those sending devices is not obscured by a proxy or firewall.
- In the case of a server-based HTTP sender, or in cases where a reverse proxy / firewall makes all HTTP requests to appear to be from a single source IP address, the IP hashing algorithm is insufficient. In such cases, this algorithm can be overridden by populating an affinity query parameter in the incoming HTTP requests.
    - The affinity parameter contains the IP address of the Avaya Breeze® platform server.
    - The IP address does not have to be addressable by the sending client. It only has to be addressable by the Avaya Breeze® platform server that hosts the load balancer.

  An example URL with affinity parameter: https://myBreezeCluster.example.com/services/myBreezeService/resource?affinity=xxx.x.x.xx.

If you would like the client IP address to be used for load balancing instead of the last hop IP address in a series of reverse proxies being traversed, please work with your administrator to configure the Cluster Attribute “Trusted addresses for converting to use X-Real-IP for session affinity”. This will make sure that when a client makes a HTTP request to Avaya Breeze® platform via Reverse proxies like Avaya Session Border Controller for Enterprise, the X-Real-IP header gets used for load balancing instead of the Avaya Session Border Controller for Enterprise IP address for load balancing.

How to decide which TLS version to be used

Many organizations are starting to require that TLS 1.3 be the minimum TLS version that is used on their networks. This is because TLS 1.3 addresses security vulnerabilities that exist in TLS 1.0,

1.1, and 1.2. However, certain legacy components can only support TLS 1.0 or TLS 1.2, so some flexibility is required in the TLS versions that are used. The high-level algorithm employed when a snap-in is creating a SSLContext is as follows:

- In general, use the global minimum TLS version across all products that System Manager manages.
- Allow an administrator to override the global minimum TLS version with a different version on a snap-in by snap-in basis.

If your snap-in only talks to a single external entity or if all external entities can support all TLS versions, do not specify a TLS version (SSLProtocolType) when creating your security context. If you do not specify a TLS version, the SSLUtilityFactory will return:

- The minimum TLS version provisioned for your snap-in on the given cluster OR
- The global System Manager TLS version if no version was specifically provisioned for your snap-in.

In certain cases, it is necessary for a snap-in to talk to multiple external entities that have varying levels of support for TLS 1.3. In such cases, create an attribute for your snap-in that allows an administrator to indicate the TLS version that is to be used to communicate to each external entity. Administrators additionally must configure the minimum TLS version for your snap-in on the Cluster Editor page. Use the indicated version to explicitly select the required TLS version when creating an SSLContext to use with that entity. If you do not do this, the provisioned minimum TLS version is used for all SSLContexts, including those that are talking to entities that support TLS 1.3.

The following table explains the TLS versions and what value their createSSLContext() returns:

System Manager TLS version	**Snap-in TLS version**	**createSSLContext()**	**createSSLContext(TLS)**	**createSSLContext(TLS 1.3)**
SSL v3	Default	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLS v1.0	Default	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLSv1.1	Default	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLSv1.2	Default	Returns SSLContext with TLS1.2	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
TLSv1.3	Default	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
SSL v3	TLS	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLS v1.0	TLS	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLSv1.1	TLS	Returns SSLContext with TLS	Return SSLContext with TLS	Returns SSLContext with TLS1.3
TLSv1.2	TLS	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3
TLSv1.3	TLS	Returns SSLContext with TLS	Returns SSLContext with TLS	Returns SSLContext with TLS1.3

SSL v3	TLS1.3	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
TLS v1.0	TLS1.3	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
TLSv1.1	TLS1.3	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
TLSv1.2	TLS1.3	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3
TLSv1.3	TLS1.3	Returns SSLContext with TLS1.3	Throws an exception and raises an alarm	Returns SSLContext with TLS1.3

Considerations about sending outbound HTTP requests

Many snap-ins have to send outbound HTTP requests to external services running on Avaya Breeze® platform or elsewhere. One common example is cloud-based RESTful web services. The XXX sample application demonstrates how to do this. Here are some important points to note when sending outbound HTTP requests:

- Depending on the network configuration where your snap-in is deployed, you might need to support an outbound HTTP proxy. To understand how to retrieve HTTP/HTTPS proxy setting information, see “How to get the HTTP/HTTPS proxy settings”.
- The recommended client for sending outbound HTTP requests is the Apache HTTP Client which is bundled on the Avaya Breeze® platform. You can use other clients, but all documentation and samples use the Apache client.
- Most outbound invocations should use HTTPS rather than HTTP. HTTPS brings additional considerations. In order to address these considerations, use the com.avaya.collaboration.ssl.util.SSLUtilityFactory class to create an SSLContext that can then be passed to the HTTP client.

  The following table contains the code snippets of the tasks that you need to perform to create SSLContext to pass it to the HTTP client. Using this class ensures that:

  - Your snap-in uses the correct TLS version that has been provisioned by an administrator. For more information, see “How to decide which TLS version to be used”.
  - Your snap-in uses the provisioned Avaya Breeze® platform WebSphere identity certificate, if challenged for a client certificate.
  - Your snap-in uses the provisioned Avaya Breeze® platform trusted certificates to validate the identity of the server. Unlike browsers, this trusted certificate store, by default, does not include the well-known Certificate Authorities (CAs). You need to manually add all trusted CAs for your deployment.
  - The outbound HTTP messages go out the eth1 (traffic) interface rather than the eth0 (management) interface. Some customers segregate their management network, and it is not possible to access many network

services, particularly those external to the enterprise, from the management network.

Task	Code snippet
Use SSLUtilityFactory to get SSLContext.	SSLUtilityFactory.createSSLContext();

Use the platform listener to get the certificate update.	@ThePlatformListener
	public class CertificateChangePlatformListener extends PlatformListenerAbstract
	{
	private Logger logger = Logger.getLogger(getClass());
	@Override
	public final void certificateStoreUpdated()
	{
	HttpClientSingleton.INSTANCE.reset();
	}
	}

Set the SSLContext in the HTTP client.	final SSLConnectionSocketFactory sslConnectionSocketFactory = new SSLConnectionSocketFactory(SSLUtilityFactory.createSSLContext(), NoopHostnameVerifier.INSTANCE);
	final Registry<ConnectionSocketFactory> registry = RegistryBuilder.<ConnectionSocketFactory> create()
	.register("http", PlainConnectionSocketFactory.getSocketFactory())
	.register("https", sslConnectionSocketFactory)
	.build();
	final HttpClientConnectionManager cm = new BasicHttpClientConnectionManager(registry);
	client = HttpClients.custom().setConnectionManager(cm)
	.build();

Use the security module interface to establish
outbound connectivity.
	final ZephyrDM dm = (ZephyrDM) DMFactory.getInstance().getDataMgr(ZephyrDM.class);
	myFqdnOrIpAddress = dm.getMySIPEntity().getFqdnoripaddr();
	requestConfig = RequestConfig.custom();.setLocalAddress(I

	netAddress.getByName(myFqdnOrIpAddress)). build(); client = HttpClients.custom().setDefaultRequestCon fig(requestConfig) .build();

- If your snap-in reuses an HTTP client and its initial SSLContext across multiple HTTP requests, be sure to listen for identity/trust certificate changes. This ensures that any changes made by an administrator are reflected immediately in your snap-in. Otherwise, for instance, an administrator might add a new trusted CA but your snap-in would not trust that CA until your snap-in or Avaya Breeze® platform is restarted. See the PlatformListener and SamplePlatformListener class in the Avaya Breeze® platform SDK Javadoc on how to listen to certificate changes.

Properties.xml

The properties.xml file is a mandatory component of a Snap-in's Archive (.svar). It allows the snap-in to define various properties and needs including the service name, the service version and service attributes. The following list describes the elements and attributes used in the properties XML file.

<service>

This is the root element of the XML file.

Level: Root Level

Type/Scope: Mandatory. Complex element.

.name

This value defines the name of the snap-in service. It should be a customer-friendly name without acronyms or abbreviations, including spaces as needed for readability. It is used for display purposes on administrative screens.

Level: Attribute of <service>

Type/Scope: Mandatory. String. Globally unique, although the same for all versions of the same service.

Purpose: Defines the version of the Snap-in service.

.version

This value should follow the Avaya versioning standard, five numbers separated by dots major.minor.update.patch.build. The first four numbers are used to determine which version is Latest, so the format is important to those components. Subsequent versions of the snap-in service should have increasing version numbers.

Level: Attribute of <service>

Type/Scope: Mandatory. String. Unique within service.

Purpose: Defines the name of the Snap-in service.

.application

This value should match the name of the service archive, the servlet EAR and the portlet WAR. It should combine a variation of the service name and the version string. It should not contain spaces and need not be a customer-friendly name. It may be needed by the service deployer to figure out which services need to be deployed or undeployed.

Level: Attribute of <service>

Type/Scope: Mandatory. String. Globally unique.

.<smgr>

This element defines the properties of the service that are needed by SMGR.

Level: Sub-element of <service>

Type/Scope: Mandatory, even if empty. Complex element.

Purpose:

.<log_space>

This value defines the maximum size allocated for snap-in logs. After log size reaches the maximum value of 10MB, it will be rolled over to next log file.

Level: Sub-element of <smgr>

Type/Scope: Optional. Integer.

.<description>

This value should be a short description of the service for the administrator, which expands somewhat on the service name.

Level: Sub-element of <smgr>

Type/Scope: Optional, but strongly encouraged. String.

.<admin_visible>

This value specifies whether or not the service is visible to the administrator. In 99% of the cases, the value will be TRUE, which is the default, so you would generally not specify it.

Level: Sub-element of <smgr>

Type/Scope: Optional. Boolean. Defaults to TRUE.

.<orig_order>

This value is only needed for call intercept snap-ins. If null or omitted, the service will not be sequenced when a call is made.

Optional. Integer. Value need not be unique. Defaults to null.

.<term_order>

This value is only needed for call intercept snap-ins. If null or omitted, the service will not be sequenced when a call is made.

Level: Sub-element of <smgr>

Type/Scope: Optional. Integer. Value need not be unique. Defaults to null.

.<orig_group>

This value is only needed for call intercept snap-ins. If omitted, the service will not be sequenced when a call is made.

Level: Sub-element of <smgr>

Type/Scope: Optional. Integer. Defaults to null.

.<term_group>

This value is only needed for call intercept snap-ins. If omitted, the service will not be sequenced when a call is made.

Level: Sub-element of <smgr>

Type/Scope: Optional. Integer. Defaults to null.

.<attribute>

This element is used to define a service attribute. Define one for each attribute needed by your service.

Level: Sub-element of <smgr>

Type/Scope: Optional. Complex element**.name**

This value is name of the attribute used by the code to retrieve its value. It typically is in camel case and does not contain spaces.

Level: Attribute of <attribute>

Type/Scope: Mandatory. String. Unique within a service.

.<displayName>

This value is a customer friendly name to describe the attribute to the administrator, including spaces as needed for readability.

Level: Sub-element of <attribute>

Type/Scope: Mandatory. String.

.<onChangeAlertMsg>

This value specifies a warning message that will be displayed to administrator upon changing the value for attribute.

Level: Sub-element of <attribute>

Type/Scope: Optional. String.

.<helpInfo>

This value is a short description of the attribute to help the administrator decide what it is used for and how to set it.

Level: Sub-element of <attribute>

Type/Scope: Optional. String.

.<attr_order>

This value specifies the order in which the attribute will be displayed.

Level: Sub-element of <attribute>

Type/Scope: Optional. Integer.

.<scope>

This value specifies the scope of attribute. The possible combinations are: 1.Global, 2.)ServiceProfile 3.) Global,Cluster and 4.)Global,Cluster,ServiceProfile. Attribute will be

applicable for the given scope(s) as mentioned here. By default, it will be applicable for all the scopes (i.e. combination 4).

Level: Sub-element of <attribute>

Type/Scope: Optional. String.

.<group>

This value specifies logical grouping of attributes. It takes name as string and order as integer for the group as sub-element

Level: Sub-element of <attribute>

Type/Scope: Optional. Complex element.

.<validation>

This element specifies the validation rule for this attribute. It is only used for scalar attributes.

Level: Sub-element of <attribute>

Type/Scope: For each <attribute>, it is mandatory that you include either a <validation> element.

.name

This value specifies the name of the validation rule. Only two rules are supported, “anyString” or “EncryptedString”.

Level: Attribute of <validation>

Type/Scope: Mandatory. String.

.<type>

This value specifies the type of the attribute. It must be STRING of validation of “anyString” and “ENCRYPTED_STRING” for validation “EncryptedString”.

Level: Sub-element of <validation>

Type/Scope: Mandatory. Enumeration.

.<pattern>

This value specifies that this validation includes a regular expression pattern match. This pattern match will be applied in addition to the validation type specified.

Level: Sub-element of <validation>

Type/Scope: Optional. String.

.<admin_visible>

This value specifies whether this attribute is visible to the administrator. In 99% of the cases, the value will be TRUE , which is the default, so you would generally not specify this.

Level: Sub-element of <attribute>

Type/Scope: Optional. Boolean. Defaults to true.

.<admin_changeable>

This value specifies whether or not this value is changeable by the administrator. In 99% of the cases, the value will be TRUE, which is the default, so you would generally not specify this.

Level: Sub-element of <attribute>

Type/Scope: Optional. Boolean. Defaults to true.

.<global>

This value is not supported and must always be set to false.

Level: Sub-element of <attribute>

Type/Scope: Optional. Boolean. Defaults to false.

.<factory>

This element optionally specifies the default as defined by the snap-in writer, it is called the factory default value.

Level: Sub-element of <attribute>

Type/Scope: Optional. Complex element.

.<value>

This value specifies the factory default for this attribute.

Level: Sub-element of <factory>

Type/Scope: Mandatory. String.

.<cutthrough_url>

This element defines the properties of the service that are needed for a snap-in to define cut through url. (see cut through URL section)

Level: Sub-element of <service>

Type/Scope: Optional.Complex element.

.<url_display_name>

This element defines the properties of the cutthrough_url to define the display name

Level: Sub-element of <cutthrough_url>

Type/Scope: Mandatory. String.

.<url>

This element defines the properties of the cutthrough_url to define the actual url

Level: Sub-element of <cutthrough_url>

Type/Scope: Mandatory. String.

About Avaya Snapp Store

Avaya Snapp Store (https://www.devconnectmarketplace.com/marketplace/) is Avaya's e-commerce solution that allows developers to post their snap-ins for purchase by other Avaya customers to monetize their snap-in. If you would like to offer your snap-in on the Avaya Snapp Store, you must register on the store as a developer and follow the onboarding process for your snap-in. As part of the onboarding process, you must include the following in the snap-in code:

1. The Supplier ID in the snap-in. See “Obtaining the Supplier ID”.
2. Your EULA text in the snap-in. See “Adding EULA”.

Obtaining the Supplier ID

About this task

The Supplier ID is used for identifying the supplier of a particular snap-in. All snap-ins from a given supplier have the same supplier id. A supplier ID is required for all the snap-ins offered through the Avaya Snapp Store; in other cases, the supplier id is optional. Use the following procedure to define the supplier ID.

Procedure

1. To obtain a Supplier ID, register to the Snapp store as a developer.

   The Supplier ID is included in any snap-in that is to be offered on the Snapp Store.

2. Define the supplier ID by adding the service attribute to the properties.xml file of the snap-in.

   For example:

```
<attribute name="com.avaya.supplierId">
    <displayName>Supplier Id</displayName>
    <helpInfo>Example supplier id</helpInfo>
    <validation name="AnyString">
        <type>STRING</type>
        </validation>
        <admin_visible>true</admin_visible>
        <admin_changeable>false</admin_changeable>
        <factory>
                <value>SUPPLIER_ID_VALUE</value>
                <user_changeable>false</user_changeable>
        </factory>
</attribute>
```

Adding EULA

About this task

Use the following procedure to add end-user license agreement (EULA) text to a snap-in. EULA is required for snap-ins that are offered on the Snapp Store, in other cases adding EULA is optional.

Procedure

1. Place the EULA text file in the resources folder of the svar module.
2. Open the manifest.xml file that is also present in the same resources folder and add the EULA component type as follows:

```
<component type="eula" filename="eula.txt" />
```

Example of a manifest.xml file with the EULA component added:

```
<?xml version="1.0" encoding="UTF-8"?>
<service_description xmlns="http://archiveschemas.aus.avaya.com/aus"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=http://archiveschemas.aus.avaya.com/aus aus.xsd name="$
{serviceName}" version="${serviceVersion}"
sdk_version="${sdk-version}" sdk_build="${sdk-build}">
        <component type="properties" filename="properties.xml" />
        <component type="eula" filename="eula.txt" />
</service_description>
```

Bundles

Service developers may want to package multiple services in a single archive called a bundle. Services packaged in a bundle can optionally declare dependencies on one or more services. Services that are declared as dependencies may be:

- Packaged within the same bundle as the dependent service.
- Packaged within a different bundle.
- Loaded independently.

Internal and external dependencies

Snap-in dependencies can be categorized as “internal” or “external”. If a snap-in is dependent on another snap-in that is included in the same bundle, this is referred to as an “internal” dependency. If a snap-in is dependent on another snap-in that is not included in the same bundle, this is referred to as an “external” dependency.

As an example, consider a bundle that includes snap-ins S1 and D1. Snap-in S1 has dependency on snap-ins D1 and D2. Here snap-in D1 is an internal dependency, and D2 is an external dependency. Snap-in D2 can be loaded individually or it can be part of another bundle.

Bundle load

Using the Load operation from the Bundles page, one or more services packaged in the bundle can be loaded at a single time. If a snap-in with the same name and version is already loaded on System Manager, then this snap-in load will be skipped. The load operation will succeed for the other snap-ins within that bundle. If an external dependency is not yet loaded on System Manager, the bundle load operation will not fail; the external dependency can then be loaded later on the “Services” page.

Partial loading of a bundle is not supported. Either all or none of the services within a bundle will be loaded. All snap-in names and versions defined in bundle.xml in the “<p:service >” element must match with the packaged snap-ins’ manifest.xml name and version or bundle load will fail.

Bundle install

Using the Install operation, one or more services can be installed on one or more clusters at a single time using bundles. Bundle install also installs the dependencies (internal/external) on the cluster if any snap-in has defined a dependency on another snap-in and the

dependency snap-in is not yet installed on the cluster. Any dependencies are installed prior to the snap-in that declared the dependencies. This allows snap-in developers to control the sequence of snap-in installation as well.

If an external dependency is not yet loaded on System Manager then the bundle install operation will fail. If a snap-in which is part of bundle or dependency snap-in (internal/external) is already installed on the cluster with the same or later version, it will not be installed again. If any snap-in installation fails on the platform, bundle installation will be rolled back. Snap-ins which were already installed on the platform prior to bundle installation will be untouched.

Bundle uninstall

Using the Uninstall operation, one or more services can be uninstalled from one or more clusters at a single time using bundles. Bundle uninstall also uninstalls the internal dependencies on the cluster if:

- Any snap-in has defined dependency on another snap-in.
- The dependency snap-in was installed on the cluster with bundle install operation.
- No other snap-ins have a dependency on the snap-in.

Any external dependencies or individually installed snap-ins are not uninstalled with bundle uninstall.

Bundle delete

Using the Delete operation, snap-ins packaged in the bundle can be deleted at a time. If a snap-in was loaded independently from “services” page then it will not be deleted. Partial bundle deletion is not supported. Either all or none of the services within a bundle will be deleted.

Cyclic dependency

A bundle load operation will fail if snap-ins declare cyclic dependencies. For example, snap-in A depends on snap-in D1 and D1 depends on Snap-in D2 and D2 depends on A. This forms a cyclic dependency, so the bundle load operation will fail in this case.

Conflicting/Overlapping dependencies

If two bundles have conflicting/overlapping dependencies, then dependency loading will be skipped for during the second bundle load operation on the same cluster as those snap-ins were already loaded during the first bundle load operation. For example, bundle B1 has snap-in X which has a dependency on snap-in D1, and bundle B2 has snap-in Y which has a dependency on Snap-in D1. Loading of bundle B1 is followed by loading of bundle B2. In this case, loading of snap-in D1 as part of bundle B2 will be skipped.

Similarly, during the bundle installation, installation of any overlapping snap-ins will be skipped, as they were already installed as part of the first bundle. Consider bundles B1 and B2 from the example above. Installation of bundle B1 is followed by B2. In this case, installation of snap-in D1 as part of bundle B2’s install will be skipped as it was already installed as part of with B1’s installation on the same cluster.

During a bundle uninstall operation, overlapping snap-ins will be uninstalled only if none of the dependent snap-ins are in the installed state after bundle uninstall. During delete

operation, overlapping snap-in will be deleted only when last bundle is deleted which is packaging it.

Bundle feature support

Bundle feature is supported from Avaya Breeze™ Release 3.3 onwards. Bundles should only include snap-ins built with Avaya Breeze™ SDK Release 3.3 or later and can only be installed on Avaya Breeze™ Release 3.3 or higher versions.

Creating bundles

About this task

To create a new bundle, start with the AuthorizationSampleBundle in the Avaya Breeze® platform SDK sample services. Copy the AuthorizationSampleBundle contents to a new directory {bundle_name} and perform the below steps in the newly created directory.

Note:

Bundle SVARs can only be loaded from the Bundles page in System Manager. Bundles cannot be loaded through the Service Management page or loaded and installed using the Eclipse plug-in.

Procedure

1. Remove any existing SVARs present in the {bundle_name}\src\main\svars directory.
2. Copy the new SVARs to be packaged in the bundle into the {bundle_name}\src\main\svars directory.
3. To define the contents of the bundle, edit the {bundle_name}\src\main\resources\bundle.xml file to add the appropriate serviceName, serviceVersion, and svarFileName as shown below.
   a. To package a service without a dependency in a bundle, add the below snippet to bundle.xml:

```
<p:services>
    <p:service name="sampleServiceName"
    version="sampleServiceVersion"
    svarFileName="sampleSvarFileNameWithDotSVARWithEx
    tension">
    </p:service>
    </p:services>
```

Here the name and version in the Bundle tag specify the name and version of the Bundle artifact that is built.There are three parts to the service tag.
Name:
The name of the service as defined in the manifest.xml of the service.

Version:
The name of the service as defined in the manifest.xml of the service.

svarFileName:

The name of the snap-in artifact / svar name that is built.
This is the name of the svar artifact that is copied to the svars directory of the bundle project

b. Package Service with dependency service packaged within the bundle.

The service should explicitly define a dependency on another service through bundle.xml using the <p:dependsOn> tag. Here the dependency service is required to be in the same bundle.

```
<p:services>
        <p:service name="sampleServiceName"
        version="sampleServiceVersion"
        svarFileName="sampleSvarFileNameWithDotSVARWithEx
        tension">
            <p:dependsOn name="dependencyServiceName"
version="dependencyServiceVersion" />
        <p:service>
        <p:service name="dependencyServiceName"
        version="dependencyServiceVersion"
            svarFileName="dependencyServiceSvarFileNameWith
        DotSVARWithExtension">
</p:service>
</p:services>
```

Here, the 'dependsOn' tag has the information about the snap-in it depends on.
This component has two parts:

- *name*
  This is the name of the snap-in in the manifest.xml of this snap-in artifact.
  When the snap-in is created using the Avaya Breeze service archetype, the manifest.xml obtains the name as the value of variable from the tag – serviceName *from the pom.xml of the main snap-in project.*
- *version*
  This is the version of the snap-in in the manifest.xml of this snap-in artifact.
  When the snap-in is created using the Avaya Breeze service archetype, the manifest.xml obtains the version as the value of variable from the tag – serviceVersion *from the pom.xml of the main snap-in project*

Here the name of the actual svar of the' depends On' snapin can be different than serviceName-version.svar.

c. Package Service with dependency service not packaged within the bundle – the dependency service can be packaged in another bundle or it is an independent service that is loaded from the services page.

The service should explicitly define dependency on another service through bundle.xml using the <p:dependsOn> tag. Here packaging dependency service in bundle is not required.

```
<p:services>
    <p:service name="sampleServieName" version="sampleServiceVersion"
    svarFileName="sampleSvarFileNameWithDotSVARWithExtension">
<p:dependsOn name="dependencyServiceName"
version="dependencyServiceVersion" />
</p:service>
</p:services>
```

Here, the 'dependsOn' tag has the information about the snap-in it depends on.
This component has two parts:

- *name*
  This is the name of the snap-in in the manifest.xml of this snap-in artifact
  When the snap-in is created using the Avaya Breeze service archetype, the manifest.xml obtains the name as the value of variable from the tag – serviceName *from the pom.xml of the main snap-in project*
- *version*
  This is the version of the snap-in in the manifest.xml of this snap-in artifact
  When the snap-in is created using the Avaya Breeze service archetype, the manifest.xml obtains the version as the value of variable from the tag – serviceVersion *from the pom.xml of the main snap-in project*

Here the name of the actual svar of the dependsOn snapin can be different than serviceName-version.svar.

d. Packaging EULA in bundle:

A bundle can contain a EULA which administrator will need to Accept to load it successfully on System Manager. If such a bundle contains individual snap-ins that have their own EULA, these individual EULAs are not displayed to the user.

To add a EULA to a bundle, follow below steps

Add eula.txt in {bundle_name}/src/main/resources. Add component type "eula" in {bundle_name}/src/main/resources/manifest.xml as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<service_description xmlns="http://archiveschemas.aus.avaya.com/aus"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://archiveschemas.aus.avaya.com/aus
../../../target/dependency/aus.xsd"
```

```
        name="${bundleName}" version="${bundleVersion}"
        sdk_version="${CE-SDK-Version}" sdk_build="${CE-SDK-
Release}">
        <component type="bundle" filename="bundle.xml" />
        <component type="eula" filename="eula.txt"/>
</service_description>
```

Update {bundle_name}/src/main/assembly/dist.xml to package eula.txt in bundle SVAR as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly
      xmlns="http://maven.apache.org/plugins/maven-assembly-
plugin/assembly/1.1.2"
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-
      plugin/assembly/1.1.2 http://maven.apache.org/xsd/assembly-1.1.2.xsd">
      <id>svar</id>
      <formats>
            <format>zip</format>
      </formats>
      <fileSets>
      <fileSet>
      <directory>src/main/svars/</directory>
      <outputDirectory>/</outputDirectory>
            <filtered>false</filtered>
      </fileSet>
      <fileSet>
            <directory>${project.build.directory}/tmp</directory>
            <outputDirectory>/</outputDirectory>
            <filtered>false</filtered>
      </fileSet>
      </fileSets>
      <files>
            <file>
               <source>src/main/resources/manifest.xml</source>
               <outputDirectory>/</outputDirectory>
               <filtered>true</filtered>
            </file>
            <file>
                  <source>src/main/resources/bundle.xml</source>
                  <outputDirectory>/</outputDirectory>
                  <filtered>true</filtered>
            </file>
            <file>
                  <source>src/main/resources/eula.txt</source>
                  <outputDirectory>/</outputDirectory>
                  <filtered>true</filtered>
            </file>
      </files>
      <includeBaseDirectory>false</includeBaseDirectory>
      <baseDirectory>/</baseDirectory>
```

```
<dependencySets>
        <dependencySet>
            <outputDirectory>/</outputDirectory>
        </dependencySet>
</dependencySets>
    </assembly>
```

4. *To change the name and version of the Bundle,* modify the {bundle_name}\pom.xml and replace values of the tags bundleName and bundleVersion in snippet below.

```
<properties>
    <bundleName>SampleBundle</bundleName>
    <bundleVersion>3.3.0.0.0</bundleVersion>
    <CeSdk.version>3.3</CeSdk.version>
</properties>
```

5. Once all of the above steps are complete, run the command “mvn clean install” to build the bundle SVAR artifact. Once this operation is completed, the artifact named bundleName-bundleVersion.svar i.e. SampleBundle-3.3.0.0.0.svar will be created in the { SampleBundle} \target directory.

Defining dependencies on pre-loaded connectors on Avaya Breeze® platform Element Manager

Procedure

1. To define a dependency on Email Connector, add the below line to the bundle.xml under entry for appropriate snap-in in the service tag:

   ```
   <p:dependsOn name="EmailConnector" version="<ServiceVersionNumber>"/>
   ```

2. To define a dependency on Scopia Connector add below:

   ```
   <p:dependsOn name="ScopiaConnector" version="<ServiceVersionNumber>"/>
   ```

3. To define a dependency on Zang SMS Connector add below:

   ```
   <p:dependsOn name="ZangSmsConnector" version="<ServiceVersionNumber>"/>
   ```

4. To define a dependency on Authorization service, add below:

   ```
   <p:dependsOn name="AuthorizationService" version="<ServiceVersionNumber>"/>
   ```

Workflows and tasks in a SVAR bundle

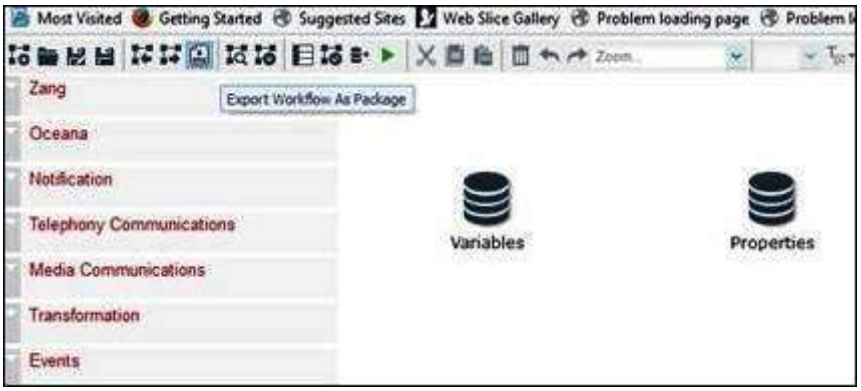
Creating a Workflow SVAR

About this task

Engagement Designer 3.9 allows users to export existing workflows from “Engagement Designer console” using the “Export Workflow As Package” button. This will create a SVAR containing the workflow. This workflow SVAR can then be packaged in a Bundle SVAR.

Procedure

On the Engagement Designer Admin console, export the existing workflows using **Export Workflow As Package**.



Result

The system creates a SVAR containing the workflow.

Creating a task SVAR

About this task

Engagement Designer allows users to export selected tasks that have previously been uploaded into Engagement Designer Administration Console on the “Bundles” tab.

To export a bundle of Tasks from Engagement Designer, navigate to the Bundles tab on the Engagement Designer Administration Console, select the task bundle, and perform the “Export to SVAR” operation. This will create a SVAR containing with the selected tasks which can then be packaged into a Bundle SVAR.

Note:

Engagement Designer Administration Console uses the term “Bundle” to refer to the package.
with a set of Tasks. This is different than the broader concept of a Bundle in System Manager.

Procedure

1. On the Engagement Designer Admin console, select the task from the Bundles list.
2. Export selected tasks using the **Export to SVAR** option.

Result

The system creates a SVAR containing the selected tasks.

Creating a bundle with the workflow and task SVARs using a sample bundle

About this task

Use the AuthorizationSampleBundle sample in the Avaya Breeze® platform SDK to create the bundle with the workflow and task SVARs.

The Eclipse plug-in does not support bundle installation. The workflow and task SVARs can only be loaded from the Bundles tab. The WFDs and Tasks SVAR must define the dependency on Engagement Designer Release 3.3 or later through bundle.xml using the dependsOn tag.

Procedure

1. For creating a new bundle using “AuthorizationSampleBundle” present in Avaya Breeze® platform SDK sample services, copy “AuthorizationSampleBundle” to new directory {bundle_name}and perform the below steps in the newly created directory.
2. Remove the existing SVARs in the {bundle_name}\src\main\svars directory.
3. Copy the workflow and task SVARs into the {bundle_name}\src\main\svars directory.
4. Modify the {bundle_name}\src\main\resources\bundle.xml file and add the metadata in the metadata.xml as shown in the following code snippet:

    - serviceName
    - serviceVersion
    - svarFileName

```
<p:services>
    <p:service name="wfdServieName" version="wfdServiceVersion"
            svarFileName="wfdSvarFileNameWithDotSVARWithExtension">
            <p:dependsOn name="EngagementDesigner" version=" 3.3.0.0.3750" />
    </p:service>
    <p:service name="taskServieName" version="taskServieVersion"
            svarFileName="taskSvarFileNameWithDotSVARWithExtension">
            <p:dependsOn name="EngagementDesigner" version=" 3.3.0.0.3750" />
    </p:service>
</p:services>
```

5. Modify the {bundle_name}\pom.xml file and add the bundleName and bundleVersion as shown in the following code snippet:

```
<properties>
    <bundleName>WFDandTaskBundle</bundleName>
    <bundleVersion>3.3.0.0.0</bundleVersion>
    <CeSdk.version>3.3</CeSdk.version>
</properties>
```

6. Run the following command to build the bundle SVAR artifact: mvn clean install

Result

The system creates the bundleName-bundleVersion.svar file, such as the WFDandTaskBundle-3.3.0.0.0.svar file, in the {bundleName}\target directory. This must be loaded from Bundles page on System Manager.

Chapter 4: Avaya Breeze® platform Call Handling

There are multiple ways that your snap-in can be involved in call handling:

- Call Intercept provides the ability for a snap-in to intercept all calls FROM a Calling Party or TO a Called Party so that application logic can be applied before the call reaches its intended destination.
- Outbound Calling starts an outbound call to a single individual and interacts with the individual directly. The snap-in can also perform the function of a broker in a call between two individuals.
- Callable Services receives an inbound call and interacts with the caller, then add an additional participant to the call.

Avaya Breeze® platform manages calls in the following way:

- Each Avaya Breeze® platform snap-in is included in a call at a signaling level independently of the others. If there are multiple Call Intercept snap-ins in the same service profile, each snap-in will each intercept the call and will get their own independent set of events. Sometimes, an action taken by one snap-in can cause another snap-in to be invoked. For example, if an Outbound Calling snap-in initiates a call to a number that is associated with a Call Intercept snap-in, that Call Intercept snap-in will be invoked. When the call is answered, both the Outbound Calling and the Call Intercept snap-in will get callAnswered callbacks.

- The same snap-in can be involved in a single call multiple times. If the snap-in acts as an Outbound Calling snap-in and that same snap-in is a Calling Party Call Intercept snap-in, it will get two callbacks for events on that call, one for each of its invocations in the call.

You can monitor and control calls without having a CallListener by using one of the call handling snap-ins. A Call Event and Control snap-in is involved in all calls where Avaya Breeze® platform is sequenced by Session Manager. This service publishes call events on the Avaya Breeze® platform Eventing Framework. Snap-ins that monitor these events or determine the UCID of calls using another method can create a Call object from the CallFactory using the App ID or UCID. The snap-in can then invoke any of the operations on the Call object that the Call Event and Control snap-in performs.
It is not possible to invoke any MediaService operations unless your snap-in was involved in a call as a Call Intercept, Outbound Calling, or Callable snap-in.

Call Intercept

Inbound call blocking

To block inbound calls with a service, in your implementation of CallListenerAbstract, inspect the call to determine if it should be allowed or blocked. Call either the allow or drop method on the Call object provided by the framework. Note that this service should be invoked on inbound calls to users for whom the service is enabled, which is also called party service. Information on configuring services to be invoked on incoming or outgoing calls is covered in more detail later in this document.

Example

```
@Override
public final void callIntercepted(final Call call) {
        if(isCallAllowed(call)) {
                call.allow();
        }else {
                call.drop();
             }
}
private final boolean isCallAllowed(final Call call) {
        // in this example we'll block calls from a specific handle
        return !call.getCallingParty().getHandle().equals("+15553091337");
}
```

Outbound call blocking

Blocking outbound calls is similar to blocking inbound calls. Examine the attributes of the call and call either the allow or drop method. Note that in this example we're looking at the called party rather than the calling party in isCallAllowed. This service should be a calling party service. It should be invoked on outgoing calls made by users for whom the service is enabled.

Example

```
@Override
public final void callIntercepted(final Call call) {
    if (isCallAllowed(call))
    {
        call.allow();
    }
    else
    {
        call.drop();
    }
}
private final boolean isCallAllowed(final Call call)
{
    // in this example we'll block calls from a specific handle
    return !call.getCalledParty().getHandle().equals("+15553091337");
}
```

When the Avaya Breeze® platform API Call drop() method is invoked before a call has been answered, Avaya Breeze® platform generates and sends a block message to the caller's endpoint. In general, endpoints can handle Avaya Breeze® platform's block message in different ways. Some endpoints might play a tone, such as reorder, while other endpoints might not play a tone or drop the line immediately.

Outbound caller ID change

First, get the Participant object that represents the caller from the Call object. Then use its setPresentedDisplayName method to change the display name that the called party will see. Then use the allow method to allow the call to exit the service. This service is a calling party service, since it is invoked for calls made by users for whom it is enabled.

Example

```
@Override
public final void callIntercepted(final Call call) {
        call.getCallingParty().setPresentedDisplayName("New display name");
        call.allow();
}
```

Redirect call

To redirect a call for either a calling or called party service, use the divertTo() method with a new destination on the Call object in the callIntercepted() method.

Example

```
@Override

public final void callIntercepted(final Call call)

{call.divertTo("18005551212"); }
```

Suggested formats of the new destination include:

- A simple telephone number that matches your system's dial plan: "18005551212"
- An E.164 number (include + before the country code) that matches your system's dial plan: "+18005551212"

  For an explanation of E.164, see https://en.wikipedia.org/wiki/E.164.

Alternate formats of the new destination include:

- A handle of a URI where the domain of the URI is the same as the domain of the original destination: "joe.smith"
- A URI with the form "handle@domain" where the handle is a name, telephone number, or E.164 number and the domain is either a host or a domain name. These forms must be used if the domain of the diverted to destination differs from that of the original destination.
  - "joe.smith@avaya.com"
  - "18005551212@destinationdomain.com"
  - "+18005551212@destinationdomain.com"

Calling Party vs. Called Party

It is possible to determine in code whether the service has been invoked for the calling party, or for the called party. The Call object provides an easy way to do this:

Example

```
@Override
public final void callIntercepted(final Call call) {
        if (call.isCallingPhase() {
                // invoked for calling party
        }else if (call.isCalledPhase() {
                // invoked for called party
        }
}
```

Zang-enabled Avaya Breeze® specific information

The CallProperties.getCallProvider(Call) is available to determine the call type (e.g., SIP or ZangCallProvider). In this environment, all snap-ins are invoked as Callable snap-ins. This means that:

- Call.wasServiceCalled() will always return true
- Call.isCalledPhase() will always return false
- Call.isCallingPhase() will always return false

Outbound calling

The Avaya Breeze® platform API provides a way for a service to initiate outgoing calls. A 1-party call could be used, for example, to ring a party and play a notification announcement. A 2-party call could be used, for example, for a click-to-call scenario.

This functionality is provided in the Collaboration Call API that is part of the larger Avaya Breeze® platform API. In particular, the CallFactory includes these methods:

- Call create(final Participant callingParty, final String target): This will create a Call which will ring the “target”. The “callingParty” is not a true party, but rather makes the call to “target” appear to be from “callingParty”.
- Call create(final String from, final String to, final Identity onBehalfOf): This will create a call between the “from” party and the “to” party. The “to” party is called first, and the call to the “from” party appears to be from the “onBehalfOf” party (but note that “onBehalfOf” is an Identity). Once the “from” party has answered, the “to” party will be called, and the call to the “to” party also appears to be from “onBehalfOf”. When the “to” party answers, the two parties will be talking.

It is a 2-step approach to make an outgoing call. First, the call is created, then, the call is initiated.

Here is a simplified example:

```
Identity onBehalfOf = IdentityFactory.create("771234");

Call twoPartyCall = CallFactory.create("779999", "775555", onBehalfOf);

twoPartyCall.initiate();
```

In this example, 779999 would ring first. When 779999 answers, 775555 would ring. When 775555 answers, 779999 and 775555 would be talking together.

Note:

An Avaya Aura® Media Server is required to make outgoing calls.

Zang-enabled Avaya Breeze® specific information

The CallProperties.getCallProvider(Call) is available to determine the call type (e.g., SIP or ZangCallProvider)

A 2-party make-call operation is not supported in a Zang-enabled Avaya Breeze® environment. See CallFactory.create(String, String, Identity) in the Avaya Breeze® platform SDK Javadoc for more details.

Participant tracking

The introduction of the ability to subtract and add participants, enabled by the new Call Termination Policy, creates the ability for a service writer to change the participants in a call in ways not previously possible. The original calling and called parties may not be involved in a call after certain changes. New methods and operations have been created to allow the service writer to obtain access to the changed participants.

The existing methods in the Call API, getCallingParty() and getCalledParty(), will now always return the original calling and called participants in the call, even if they are no longer active. The participants currently active in a call can be retrieved using the method getActiveParties(), which returns a list of Participants. The display information for these participants can be updated with the three existing methods setPresentedHandle(), setPresentedDomain() and setPresentedDisplayName(). The existing method getAlertingParties still returns a list of Participants, but now those Participants can be updated with Presented information as well.

To aid in tracking a participant's activity in a call, a new method is added to the Participant API. getState() returns the ParticipantState of the Participant, which can be IDLE, ORIGINATING, ALERTING or CONNECTED. Note that the states only change when a change is confirmed. For instance, the called party is IDLE in the CallIntercepted callback because it is not yet known whether the call will be allowed. Comparably a Participant will not progress to ALERTING until a response is received to confirm that the Participant is ALERTING.

Dropping and adding participants

To drop and add participants in a call, a service can invoke dropParticipant() and addParticipant(). The first requirement to make use of these abilities is to set the Call Termination Policy to NO_PARTICIPANT_REMAINS so that a call will not drop when one

party drops out of a call or is dropped from the call and another party remains. When the policy is set this way, Avaya Aura® Media Server is required in the system. Media Server is needed to provide feedback tones to the remaining participant in a call. The default Call Termination Policy is ONE_PARTICIPANT_REMAINS, which means that when either party drops out of a call the entire call drops.

Methods

Insert content for the first section.

setCallTerminationPolicy

Example

```
final CallPolicies callPolicies = call.getCallPolicies();

callPolicies.setCallTerminationPolicy(CallTerminationPolicy.NO_PARTICIPANT_REMAINS);
```

DropParticipant

The new method dropParticipant can be invoked on any currently active participant in a call. If invoked when only one participant is active the entire call will be dropped. If two or more participants are active the specified participant will be dropped and depending on the resulting state, all participants may still be dropped. For example, when a call is ringing dropping the calling party will drop the entire call. When a participant drops or is dropped a new participantDropped callback will be invoked. Note that if dropping a single participant leads to dropping both participants the callback will be invoked twice (once for each participant), followed by the callTerminated callback.

AddParticipant

The existing method addParticipant() that was previously used to add additional target parties during the alerting phase can now also be used to add a participant to a call that currently only has one participant remaining. It can still be used to achieve parallel ringing. If a call is starting from a two-party state and the service wishes to drop one participant and add another one, it is necessary to wait for the participantDropped() callback after dropping the first participant before the addParticipant() method can be invoked. If the attempt to add a participant fails a new callback is invoked, addParticipantFailed(). If the extra party is added without issue the existing callAlerting callback will be received.

Note:

The addParticipant method can only be used to go beyond 2 parties when the call is alerting (i.e., for parallel forking). After the call has been answered, it is not possible to add more than 2 parties and therefore this method cannot be used to create conference calls.

Examples

Sequential Ringing: This feature refers to the new ability to attempt to ring a called party, then drop that participant after a timer expires and attempt to ring another party. This can be done by calling Call.allow, then after a timeout call call.dropParticipant(call.getCalledParty()), followed by call.addParticipant(“18005551212”). The call to addParticipant should not be invoked until the participantDropped callback is invoked.

Serial Calling: This feature refers to the new ability to make a series of calls on behalf of the calling party. A caller could request to speak to Bob and then Carol and the snap-in can

detect when Bob drops out and use the participantDropped callback to addParticipant("Carol").

Zang-enabled Avaya Breeze® specific information

The CallProperties.getCallProvider(Call) is available to determine the call type (e.g., SIP or ZangCallProvider)

There are restrictions pertaining to adding a participant to a Zang call. See Call.addParticipant(Participant) in the Avaya Breeze® platform SDK Javadoc for more details.

The feature option of keeping a call active when a participant hangs up from a two-party call is not supported in this environment, which means that the NO_PARTICIPANT_REMAINS option is not available.

Flexible Call Leg Control

Multiple call targets

To add an additional called party in parallel with the original called party use the addParticipant() method with a new destination. The method is invoked on the call object in the callIntercepted() method. The added participant will only be added if the service allows the call to proceed using either allow() or divertTo(), and only after the called or diverted party is ringing. If a party cannot be added the addParticipantFailed callback will be invoked.

Example

```
@Override
public final void callIntercepted(final Call call)
{
call.addParticipant("18005551212");
call.allow();
}
```

See the section on Redirect call for information about formats of the destination.

Callable Service

What is a callable service?

Callable Services are services that are invoked as a result of being directly called, as opposed to being invoked on the behalf of an end user when that user originates or receives a call. An example would be a snap-in that is handling incoming calls to a contact center's 800 number.

Callable service snap-in configuration

In a pure call intercept scenario, Avaya Breeze® platform is provisioned as a "sequenced application" that is invoked by Session Manager in the originating phase (invoked on the behalf of a caller) and/or the terminating phase (invoked on the behalf of the called party). Avaya Breeze® platform is able to invoke the appropriate snap-in(s) based on the provisioned service profile for the user.

In a callable service scenario, Avaya Breeze® platform is not provisioned as a sequenced application. Instead, a routing policy is configured to route calls with a particular pattern to Avaya Breeze® platform. Session Manager sends the call there without an expectation that the call will be coming back so that it can be sent on to its end destination. This is the key differentiation between a call intercept and callable snap-in.

You can invoke Call Intercept Snap-ins before sending a call to a Callable Snap-in. In this case, Session Manager is not configured to send the call to Avaya Breeze® platform as a sequenced application. The same routing policy configuration is used as for a standalone Callable Service, except multiple snap-ins are configured in the Service Profile. The last snap-in in the sequence is treated as a Callable Service, and the others are treated as Called Party Call Intercept Services.

More details on how to configure a snap-in as a callable service can be found in *Administering Avaya Breeze® platform*.

How to write a snap-in that acts as a callable service

A snap-in does not need to define any special tags or attributes to make it a callable service. The difference is entirely in the Session Manager configuration.

A typical callable service might do some of the following in its callIntercepted() callback implementations:

- The logic to divert a call to a different destination based on certain conditions.
- The logic to play an announcement and ask for digits to the caller, then take further action based on entered digits.

Though it is unlikely, it is possible for a snap-in to act both as a call intercept snap-in and a callable service. Therefore, Avaya Breeze® platform provides an API for a snap-in to identify whether it was invoked as a callable service: Call.wasServiceCalled() API has this functionality. More details on this API can be found in the SDK Javadocs. This could be important since things work slightly differently for callable snap-ins versus call intercept snap-ins. We strongly recommend that you use this method.

“com.avaya.collaboration.call.sample. SampleCallableService” is an example of call listener for a callable service. Javadocs for this class can be found in the SDK.

Avaya Breeze® platform API differences for a callable service

When a snap-in is invoked as a callable service, a few APIs behave differently than they do for call intercept snap-ins.

- Call.allow() API – A snap-in generally invokes Call.allow() when it wants the call to proceed to the original target. When a snap-in is configured as a callable service, it

itself is the original target of the call, hence it does not need to invoke call.allow() to proceed with the call. This operation has no effect in a callable service scenario.

- Call.addParticipant(Participant Party) API – When a snap-in invokes Call.addParticipant in call intercept scenario, the added participant starts ringing only when original called party starts ringing. When a snap-in is configured as a callable service, the addParticipant operation takes effect immediately and invocation of the allow() or divertTo() API is not required.

Call.wasServiceCalled() API: This API returns true if the service was invoked as a Callable service or false if invoked as a Calling Party or Called Party service. We strongly recommend that you use this method in your callable service.

Inserting and removing Avaya Aura® Media Server

Avaya Breeze® platform can add the Avaya Aura® Media Server into the media stream automatically whenever a media operation is performed. Most snap-ins will not have to give any thought as to when or how to insert Media Server; it will just happen when a media operation is invoked. This is true both before the call has been answered (during the callIntercepted() callback) and after the call has been answered. Similarly, most snap-ins need not be concerned about removing Media Server after the media operation completes. After several seconds go by without any active media operations, Media Server will automatically be removed. The list of media operations that will cause Media Server to be inserted include play, record, send or collect digits, speech search, and Voice XML dialog.

The default Media Server inclusion policy of “AS_NEEDED” can be overridden to instead be set to “INCLUDED”. With this setting, Media Server will immediately be added to the call and will not be removed until the setting is changed to “AS_NEEDED”. There are two cases where a snap-in will want to use the “INCLUDED” Media Server inclusion policy:

1. The process of inserting and removing Media Server causes a momentary disruption in the audio path. In some cases, a snap-in may find it preferable to leave Media Server in the flow for the duration of the call rather than having these disruptions.
2. There is a window of time in which Media Server cannot be inserted: after a snap-in invokes allow()/divertTo() and before it receives the answered() callback. If a snap-in doesn’t need to invoke a media operation in the callIntercepted() callback but wants to reserve the right to do so before the call is answered, it should set the policy to “INCLUDED” during the callIntercepted() callback. It can then set the policy back to “AS_NEEDED” after the call is answered.

Note:

The enableMediaBeforeAnswer has been deprecated and should no longer be used.

Methods

setMediaServerInclusion:

Example:

```
final CallPolicies callPolicies = call.getCallPolicies();

callPolicies.setMediaServerInclusion(MediaServerInclusion.INCLUDED);
```

Insert Media Server MediaServerInclusion.INCLUDED:
This changes the policy immediately and causes Avaya Breeze® platform to not remove the media stream unless the media policy is set to AS_NEEDED. If this was invoked after the call has been answered, Media Server will be inserted when the next media operation is invoked.

Remove Media Server MediaServerInclusion.AS_NEEDED:

This default policy causes Avaya Breeze® platform to remove Media Server from the stream when no media operations are active. If the previous policy was INCLUDED, Media Server would be removed immediately if there were no media operations active at the time of invocation.

Media operations on mixed audio stream

To play an announcement or start speech search on a call, the service can specify the API method with a call parameter in its signature. The method with this signature specifies the mixed audio stream as the target on which the Avaya Aura© Media Server should invoke the media operation. This means that when play announcement is invoked in the callIntercepted method, the calling party will hear the full announcement. However, the called party will only hear the portion of the announcement that is remaining after answering the call. Also, this means that when start speech search is invoked in the callIntercepted method, the calling party's speech will be analyzed for the duration of the call. However, the called party's speech will be analyzed only after answering the call.

Methods

play, startSearch

Example

final UUID requestId = mediaService.play(Call call, PlayItem playItem, MediaListener mediaListener);

final UUID searchId = speechService.startSearch(Call call, SearchOptions searchOptions, SpeechSearchListener speechSearchListener);

Service invocation configuration

Avaya Breeze® platform currently supports three modes of service invocation: calling party services, called party services, and callable services.

- A calling party service is invoked when a user with the service enabled in their Service Profile makes a call.
- A called party service is invoked when a user with the enabled service in their Service Profile is called.
- For callable service invocation, see the “Callable Services” chapter. The example properties.xml file for callable services is same as the configuration for called party services.

Configuring your service as either a calling or called party service is done by editing the properties.xml file in the resources directory of the SVAR project (testService-svar/src/main/resources with the values from the example above).

Note that a calling party service includes the orig_order and orig_group elements. The values provided for these are irrelevant, but some value must be present (you could simply use “1” for both). The following is an example of a minimal properties.xml for a calling party service:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns=http://archiveschemas.aus.avaya.com/properties
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://archiveschemas.aus.avaya.com/properties properties.xsd"
    name="${serviceName}" version="${serviceVersion}" application="${serviceName} -
    $ {serviceVersion}">
        <smgr>
            <description>Test Service</description>
            <orig_order>1</orig_order>
            <orig_group>1</orig_group>
            <fs_component>true</fs_component>

        </smgr>
</service>
```

Note that a called party service includes the term_order and term_group elements. The values provided for these are irrelevant, but some value must be present (you could simply use “1” for both). The following is an example of a minimal properties.xml for a called party service:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns=http://archiveschemas.aus.avaya.com/properties
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://archiveschemas.aus.avaya.com/properties properties.xsd"
    name="${serviceName}" version="${serviceVersion}" application="${serviceName} -
${serviceVersion}">
    <smgr>
        <description>Test Service</description>
        <term_order>1</term_order>
        <term_group>1</term_group>
        <fs_component>true</fs_component>
    </smgr>
</service>
```

Removing service from call

This feature allows a Snap-in to request that Avaya Breeze® platform be removed from the call signaling path without the call being dropped. This helps conserve Avaya Breeze® platform resources.

Snap-in Implementation

If the snap-in is a call intercept or two party make call snap-in, the implementation can invoke ServiceManager.removeServiceFromCall(call) in any state. If it is invoked before the call becomes stable with 2 parties, the request will be queued.

Example:

1. Snap-inX receives call (Call Intercept)
2. Snap-inX invokes removeServiceFromCall()

```
@Override
public final void callIntercepted(final Call call)
{
        ServiceManager.removeServiceFromCall(call);
}
```

3. Avaya Breeze® 'queues' the remove service from call request until the called party answers.
4. Snap-inX allows the call to the Called participant.
5. When the called party answers, Snap-inX will be removed from the signaling path. In case Snap-inX is the only snap-in in the application sequence, Avaya Breeze® platfom is removed from the signaling path.

If the snap-in is invoked as a callable snap-in, the implementation should invoke ServiceManager.removeServiceFromCall(call) only after invoking addParticipant() or divertTo().

Example1:

1. Snap-inX receives call (Callable Service)
2. Snap-inX invokes removeServiceFromCall()
3. Avaya Breeze® platform will throw an IllegalStateException because Avaya Breeze® platform only knows of one party on the call (Calling participant)

Example2:

1. Snap-inX receives call (Callable Service)
2. Snap-inX invokes addParticipant() or divertTo()
3. Snap-inX invokes removeServiceFromCall()
4. Avaya Breeze® platform 'queues' the remove service from call request until the called party answers.

5. When the called party answers, Snap-inX will be removed from the signaling path. In case Snap-inX is the only snap-in in the application sequence, Avaya Breeze® platform is removed from the signaling path.

ServiceManager.removeServiceFromCall(call) cannot be used if there is a single party on a call without a pending action to add a second party. The method must only be invoked after a second party has been added to the call by invoking addParticipant(). Note that this condition does not apply to call intercept and two-party make call, as both of these scenarios have a pending action to add a second party.
There are two ways that a snap-in can be involved in a one-party call:

- The snap-in initiated a one-party make call
- The snap-in set the call termination policy to NO_PARTICIPANT_REMAINS and only one party remains in the call.

Service listener implementation

It may be important for the snap-in to know whether the invocation to ServiceManager.removeServiceFromCall succeeded. In order for a snap-in to be notified of the outcome, the application must create a serviceListener class, annotated with TheServiceListener and implementing the ServiceListener interface as shown below:

```
@TheServiceListener
public class SnapinServiceListener implements ServiceListener
{
      public void removeServiceFromCallSucceeded(Call call)
      {
            .....
      }
      public void removeServiceFromCallFailed(Call call, String
      reason)
      {
            .....
      }
}
```

Chapter 5: Developing the service to use the Cluster DB

Introduction to Cluster DB

The Cluster DB provides a relational database (DB) on a per cluster basis that snap-ins can easily set up and use. This section focuses on what a developer needs to do to create and access the DB from the snap-in. The snap-in can then access the DB using JDBC, Open JPA, or Hibernate. The snap-in can create and upgrade the schema from snap-in version to snap-in version using the schema upgrade feature. The snap-in can also get a JDBC Datasource setup with a connection pool to access it DB using JNDI.

Note:
It is not recommended to perform database operations, especially write operations, in a SIP call thread. See the Performance and scalability considerations chapter on how to move to a non-SIP thread for database operations.

Database setup

A snap-in can setup one or more databases instances by adding a database section in the properties.xml file. Each database instance can have its own set of tables and structure and is independent from the other database instances contained in the Cluster DB. In this section, one DB instance in the Cluster DB that the snap-in uses is referred as “DB”. You can specify the following attributes for a DB:

Name	Default	Example	Required	Description
Name	–	Mydb	Yes	A short name that is appended to your snap-in name with underscores in between words.
container-mgmtname	–	jndi</mydb	No	The JNDI name of the datasource. This name is automatically created and you must specify this name in the persistence.xml file.
max-conns	10	8	No	The maximum number of connections in the connection pool. Only used if container-mgmt-name is specified. You this attribute sparingly since connections are a limited resource and once specified each node in the cluster uses these many connections.
non-transactional	False	True	No	If you do not want container managed transactions, set this attribute to True. Only used if containermgmt-name is specified.
multi-threadedaccess	False	True	No	If you want to check the multi-threaded access flag in the datasource, set this attribute to True.

Description	–	Call Statistics	Yes	To provide more information about the attribute.
schema-major	–	0	No	Not required for the current version. However, required for version 3.1, so you must set this attribute to 0, even if it is not used.
schema-minor	–	0	No	Not required for the current version. However, required for version 3.1, so you must set this attribute to 0, even if it is not used.
schema-revision	–	0	No	Not required for the current version. However, required for version 3.1, so you must set this attribute to 0, even if it is not used.

Properties.xml

As an example, we can use SampleSnap as our snap-in. To set up a DB, the Properties.xml file for SampleSnap can include the following attributes:

```
<database>
        <instance>
                <name>statdb</name>
                <container-mgmt-name>jdbc/statdb</container-mgmt-name>
                <non-transactional>false</non-transactional>
                <max-conns>5</max-conns>
                <description>Holds test statistics</description>
        <</instance>
</database>
```

Ensure that the properties.xml file contains Unix Format (LF) End Of Line (EOL) characters of the instead of Windows Format (CRLF). For the Windows platform you can use the “dos2unix'” free utility or the “Notepad++ text editor” to do this conversion. Or on the Unix platform you can use dos2unix command. Or in an IDE (Eclipse/Netbeans) you can use plugins available with them. You can set the lineEnding attribute to unix in dist.xml section in the Properties.xml to avoid running into issues.

When SampleSnap is installed on a cluster the following takes place:

- If the DB user of SampleSnap is not present then the DB user named samplesnap is created. (Remember: Postgres does not use capitals).
- DB named samplesnap_statdb is created, if not already present. It is owned by the user samplesnap.
- A JDBC datasource that is transactional is created that has five connections in the pool and can be looked up with JNDI name of jdbc/statdb.

Persistence.xml

Following is a sample persistence.xml that displays the use of the JDBC datasource. Persistence.xml is located at SampleSnap/SampleSnap-war/src/main/resourses/META-INF/persistence.xml.

```
<?xml version="1.0"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <persistence-unit name="statdb" transaction-type="JTA">
            <jta-data-source>jdbc/statdb</jta-data-source>
                    <class>com.mycom.snapins.feature.db.Stats/class>
                                    ...
            <properties>
                    ...
            </properties>
    </persistence-unit>
</persistence>
```

The snap-in can now access and use the created DB named samplesnap_statdb. You must first create and manage your DB schema before you can use the DB.

DB schema creation and maintenance

This section explains how to create, update, and manage the tables in the snap-in DB. The schema is managed using a version number, which is independent of the snap-in version. Schema versions might not change with each new version of the snap-in since each new version of the snap-in might or might not require a schema update. Use schema upgrade scripts to initially create the schema and also for subsequently updating the schema. Schema versions consist of three positive integer number separated by a dash. The numbers represent in order, the major, minor, and revision of the schema. All snap-in schema numbers start at 0-0-0. The 0-0-0 version of the schema does not contain tables or anything else, it is just a newly created DB.

Schema upgrade scripts

Each schema upgrade script takes the schema from one schema version, the “from” version, to another schema version, the “to” version. The snap-in developer decides the from and to versions on each script. Over the lifetime of a snap-in, there might be many schema changes and therefore many schema upgrade scripts. The upgrade scripts are stored in the .svar at the top level directory named dbupgrade. The upgrade script is an .sql script containing DDL and SQL that creates new tables, alters existing tables and inserts seed data. The upgrade script does not contain transaction syntax because it is run in a single transaction. Each new snap-in version includes all the previously written schema upgrade scripts to ensure that the DB schema can be made current irrespective of the contents of the currently existing schema. The scripts are executed in a progression determined by the from and to version along with the current version of the DB. The scripts must following a naming convention that the Upgrade Engine uses to determine the from and to versions of the DB. The naming conventions are detailed in the following section.

Naming of the upgrade scripts

Following components make up the name of the upgrade scripts:

1. upgrade_
2. db name (short name taken from the properties.xml file)
3. _ (an underscore)
4. from version (FromSchemaVersion)

5. _ (an underscore)
6. to version (ToSchemaVersion
7. .sql

So, for example, upgrade_statdb_0-0-0_3-0-0.sql is the name of the script that creates the initial schema of the DB used by SampleSnap. Schema version 0-0-0 is the initial blank schema version. After this script is run and completes successfully, the current schema version is now the FromSchemaVersion or 3-0-0. The current schema version of each DB is maintained in a system table. After an upgrade script runs successfully, the current schema version is updated to the ToSchemaVersion of the script. Snap-in developers can determine the schema number. However, the progression of upgrade script execution is determined by chaining together the FromSchemaVersion and the ToSchemaVersion, so you must be careful not to break the chain. In this case, the next version of the schema upgrade script must have a FromSchemaVersion of 3-0-0.

Snap-in upgrade scripts tips

- Include a database section in the properties.xml. This sets up a DB instance with a name for the snap-in to use.
- Include all the upgrade scripts representing each time the schema has been updated over the life of your snap in. Follow the naming conventions described in “Naming of the upgrade scripts”.
- Ensure that the very first upgrade script has a FromSchemaVersion of 0-0-0. In most cases, this script creates all the tables for the first version of the snap-in.
- Schema versions are not tied to snap-in versions. There might be many schema versions between a snap-in release version or there might be none.
- Schema versioning persist throughout all versions of the snap-in. The snap-in developer can determine the version numbering.
- Upgrade scripts are run as one transaction so all of it succeeds or none of it succeeds. If the upgrade script fails the current schema version remains same. Do not include transactions control statements such as begin and commit, in your upgrade scripts.
- Schema versions must be backwards compatible with older versions of a snap-in. If you drop a column of a table, chances are you have broken backwards compatibility. The older version will fail when it tries to insert, update, or select on the column that was dropped.
- Schema versions do not have to be consecutive but it is a good idea within one version. For example, upgrade_mydb_1-1-10_2-3-22.sql..
- Ensure that you tie the ToSchemaVersion and FromSchemaVersion chain together so that version progresses to current. That is, a new upgrade script must have a FromSchemaVersion of the previous version's ToSchemaVersion. The upgrade run ends when there is no upgrade script with the FromSchemaVersion of the current schema version.
- Avoid loops in the progression of execution by making the versions go progressively higher.
- There are no downgrades, once the schema is upgraded it stays on that version, even if the new version is uninstalled or deleted.
- Older versions of the snap-ins must work with newer schema versions.
- Upgrade messages are located at /var/log/Avaya/sm/clusterdb.log, which is filtered using the `hadb_ctl printlog` command.

Notifications for Cluster DB backup and restore

Snap-ins can request to be notified when backup and restore operations occur on the Cluster Database.

Usage

A snap-in can define a listener class to get backup and restore status change notifications. The listener can be defined by implementing the interface “com.avaya.zephyr.platform.dm.DMListener”. The method “objectChanged()” gets invoked whenever there is a change in Cluster Database backup and restore status.

A sample code for this can be seen below:

```
public class TestDaoListener implements DMListener
{
	private static TestDaoListener listener = new TestDaoListener();
	private Logger logger = Logger.getLogger(TestDaoListener.class);

	private TestDaoListener()
	{
	}

	public static TestDaoListener getInstance()
	{
		return listener;
	}

	@Override
	public void objectChanged(Object oldObject, Object newObject)
	{
		if (oldObject instanceof AusBackupRestoreData || newObject instanceof
	AusBackupRestoreData)
		{
			if (newObject != null)
			{
				AusBackupRestoreData newData = (AusBackupRestoreData)
			newObject;

				logger.info("Service Name: " + newData.getServiceName());
				logger.info("Database Name: " + newData.getDbName());
				logger.info("Schema version: " + newData.getSchemaVersion());
				logger.info("Cluster Name:" + newData.getClusterName());
				logger.info("Operation: " + newData.getOperation());
				logger.info("Operation Status: " + newData.getStatus());
			}
		}
	}
}
```

The listener needs to be registered with DAO (AusBackupRestoreDAO) to get notifications. And the listener should also be removed during service uninstallation. The registration and removal of listeners can be done in the init() and destroy() methods of ServiceLifeCycle class as shown below:

```
@TheServiceLifeCycle
public class MyLifeCycleClass implements ServiceLifeCycle
{
	private Logger logger = Logger.getLogger(MyLifeCycleClass.class);

	@Override
	public void init()
	{
	// Registration of Listener with "AusBackupRestoreDAO" for notifications of backup/ restore
	status value changes

DMFactory.getInstance().getDataMgr(AusBackupRestoreDAO.class).registerListener(TestDaoList
ener.getInstance());
	}

	@Override
	public void destroy()
	{
		// Removal of Listener from "AusBackupRestoreDAO"
DMFactory.getInstance().getDataMgr(AusBackupRestoreDAO.class).removeListener(TestDaoListen
er.getInstance());
	}
}
```

Upgrade engine

The upgrade engine runs each time a snap-in is installed on a cluster for each DB instance specified in the properties.xml file. The upgrade engine looks for scripts to run and stops when it does not find a matching FromSchemaVersion, in which case it exits successfully. When a script fails the upgrade, engine quits looking and exits as failed. Steps followed by the upgrade engine steps are as follows:

Steps followed by the upgrade engine:

1. Retrieve the current version of the DB from system table, if not found use 0-0-0.
2. Look for an upgrade script with its from version equal to the current version. If the upgrade engine cannot find any then exit as a successful upgrade.
3. Run the found script as a single transaction. If it fails, everything done in script is rolled back, schema remains the same, and the upgrade engine exits as a failed upgrade.
4. Upgrade the current version of the DB in the system table.
5. Go to step 1.

Inserting the upgrade scripts into your svar

Procedure

1. Add a file section in the SampleSnap/SampleSnap-svar/src/main/assembly/dist.xml for each upgrade script.

   An example containing two upgrade scripts:

```
<file>
        <source>src/main/resources/upgrade_statdb_0-0-0_3-0-0.sql</source>
        <outputDirectory>/dbupgrade</outputDirectory>
        <filtered>true/filtered>
</file>
<file>
        <source>src/main/resources/upgrade_statdb_3-0-0_3-1-0.sql</source>
        <outputDirectory>/dbupgrade</outputDirectory>
        <filtered>true</filtered>
</file>
```

2. Place the upgrade scripts in the SampleSnap/SampleSnap-svar/src/main/resource directory.

   The maven build automatically includes the scripts in the svar.

DB removal

Once created, the DB is not removed immediately when the snap-in is uninstalled from the cluster, or even when the last remaining version of the snap-in is uninstalled from the cluster. The DB and all its data is removed after around one week of no snap-in being installed on the cluster that is using the DB. This allows for uninstalling of a snap-in for short period of time without fear of losing its data.

Accessing the Cluster DB

JPA (OpenJPA or Hibernate) can look up the datasource using the jndi name specified in properties.xml instead of using a JBDC connection.

SSL client authentication is used to ensure that only Avaya Breeze® platform nodes within the same cluster are able to access the cluster DB.

Chapter 6: Avaya Breeze® platform connectors

Avaya Breeze® platform comes pre-loaded with three connectors that can be used to send email and SMS messages and to schedule conferences. Each has an API so your services can quickly incorporate email, sms and conferencing functionality.

- Scopia Conferencing Connector to setup conferences.
- Email Connector to send Email messages.
- Zang SMS Connector to send SMS messages.

For information about Zang SMS Connector, see the *Zang SMS Connector Snap-in* document.

Scopia connector

Introduction

The Scopia Connector enables programmatic access from Avaya Breeze® platform to a Scopia Conferencing system. The connector, in combination with the Conferencing API, provides a convenient way for service writers to schedule, list, and cancel conferences using Java. The connector uses HTTP/S to communicate with the Equinox Management application through the Scopia XML-based API. Please note that the XML-based Equinox Management API is an 8.x and higher feature of Scopia systems and only enabled with certain models/configurations of Scopia systems. Please check with your Scopia representative for details.

Overview

To use the Scopia Connector, you'll minimally need to configure a couple of things in Equinox Management. You optionally can also configure Equinox Management to allow the use of HTTPS. A summary of how to configure Equinox Management appears in this section. Please refer to the Scopia documentation that came with your Scopia system for more detailed configuration information. You will also need to configure the connector in System Manager, as detailed below.

Service provider (multi-tenant) support

Scopia supports service provider deployments for multiple organizations (tenants). In a multi-tenant deployment, each Scopia meeting is associated with only one tenant and visibility across tenant boundaries is restricted.

To use the Scopia connector in a multi-tenant mode, you must:

- Enter a company name in the **Organization Name** field.
- Use the organization name as part of the user name format.

For more information, see Scopia connector field descriptions.

Configuration summary

First, you'll need to create a Meeting Type that will be used when scheduling conferences through the connector. A Meeting Type defines the audio and video resources that will be used during a conference. You'll want to make note of the prefix that gets assigned to this Meeting Type. This value will be used for later configuration of the connector in System Manager.

Second, you'll need to create a user account in Equinox Management. This account is a standard user account from the point of view of Equinox Management, but it will be the account used to authenticate API connections from Avaya Breeze® platform. To ensure the account has adequate authority to perform all of the necessary functions, you will need to create an account and assign it a profile with at least the following characteristics enabled:

- "Can schedule meetings"
- "Can invite endpoints and reserve resources"
- "Can record meetings"

You'll also need to select the Meeting Type created previously as an "allowed" meeting type for this profile.

Finally, if you would like to configure Equinox Management to allow the use of HTTPS by the SCOPIA Connector, perform the steps described in the section "Configuring the Tomcat Web Server to Use HTTPS" of the *Administrator Guide for SCOPIA Management* document.

Configuring the Scopia connector

About this task

Configure attributes for the Scopia Connector using Avaya Aura® System Manager

Procedure

1. From the Elements panel, select **Avaya Breeze®**.
2. From the **Configuration** menu, select **Attributes**.
3. Select the **Service Globals** tab; then select **ScopiaConnector** from the **Service** menu.
4. Fill out the Attributes Configuration page for the SCOPIA Connector according to the Scopia connector field descriptions.

Scopia connector field descriptions

Field	Description
Dial-in Number for Conference Access	This is the phone number used to contact the auto-attendant of your SCOPIA system. A user will typically dial this number and then enter a conference number to gain access to a conference.

Field	Description
Meeting ID Length	The number of digits used for the Meeting ID. It must be greater than or equal to the **Minimum Meeting ID Length** configured on the conference service.
Organization Name	Used only in multi-tenant deployments. The organization name configured on the conference service for scheduling conferences from Avaya Breeze® platform.
Password for API Access to Conference Service	This is the password previously configured in Equinox Management.
Service Prefix for Scheduling Conferences	This is the prefix assigned to the Meeting Type that was previously configured in Equinox Management.
Test mode enabled?	When you are first testing your service that uses the Scopia Connector (typically through the Scheduled Conference API), you might want to set this to **true** (check the **Override Default** checkbox and change the **Effective Value** to **true**). When test mode is enabled, the Scopia Connector runs through a subset of its typical behavior and then forms a typical response that is returned to the requesting service.
URI for API Access to Conference Service	This is the URI that provides access to the XML-based API of the Equinox Management application. You can use either HTTP or HTTPS. If you use HTTPS, you'll need to be sure that Equinox Management has been configured to allow TLS connections. Also, depending on the configuration of Equinox Management, for HTTP communication you may need to specify a port number of 8080; similarly, you may need to specify a port number of 8443 for HTTPS communication. It is recommended that you use HTTPS communication to ensure that the password used to authenticate to Equinox Management is secure on the network.
URI for Conference Access	This is the URI used by conference attendees to gain access to a conference using a browser.
User Name for API Access to Conference Service	This is the user ID previously configured in Equinox Management. In multi-tenant deployments, this field must be formatted to include the Organization Name. For example, for Company A, the correct format is: <*user name*>@CompanyA.

Using the SCOPIA Connector from your service

Use the Scheduled Conference (SchedConf) API (which is a part of the larger Avaya Breeze® platform API suite) to schedule, list, and cancel conferences through the SCOPIA Connector. The Scheduled Conference API is a fairly flexible API designed to operate with potentially more than just a SCOPIA system, so some of the method names in the API will not appear to directly correlate with a specific conferencing connector. The API primarily consists of a SchedConf object that you obtain from a factory and populate with the various information to schedule a conference (via the schedule() method). Other operations allow you to list and cancel conferences that have not yet started.

The following code snippet illustrates the use of the API to schedule a conference:

```
final SchedConf conf = SchedConfFactory.createConf();
```

```
// Set our subject and a duration of 4 hours. conf.setSubject("Meeting for Urgent Customer Request").setDuration(0, 4, 0); conf.setParticipantPin(-1).setModeratorPin(-1); // Generate pins for me

try
{
	conf.schedule();
}
catch (SchedConfException e)
{
	System.out.println("Error while scheduling a conference; " + e);

}

// // The getUrl() method returns the URL that participants use

// to join a conference from a browser.

//

System.out.println("URL=" + conf.getUrl());
```

This code snippet results in the scheduling of a conference that will start immediately. The API also allows the start time to be explicitly set to a future date and time.

Note that for the immediate scheduling of a conference to run reliably, the Avaya Breeze® platform host and the Equinox Management host must be tightly synchronized. This synchronization can be achieved by the use of a time server. If the servers are not synchronized, one of two things can happen:

- If the Avaya Breeze® platform host has a time later than the time on the Equinox Management host, the start time of the conference will be delayed.
- If the Avaya Breeze® platform host has a time earlier than the time on the Equinox Management host, the conference may fail to be scheduled, depending on the version of Equinox Management that is being run. Some versions of Equinox Management will reject an attempt to schedule a conference that has a start time in the past.

To ensure the successful scheduling of a conference in the absence of tight synchronization, it is suggested that a service schedule “immediate conferences” 1 or 2 minutes in the future to avoid the possibility of a schedule request being rejected by Equinox Management.

Email connector

Introduction

The Email SMTP Connector is a convenience service provided by Avaya Breeze® platform that gives a service writer an easy way to send an email to 1 or more recipients. The service writer interacts with the Email SMTP Connector using the Email API that is part of the larger Avaya Breeze® platform API suite.

As its name suggests, the Email SMTP Connector communicates using only SMTP (the Simple Mail Transfer Protocol) and SMTPS, so this connector can be used only to send email. You do not need to buy a license for Email Connector to work. However, to connect to any licensed email server you must add a trust certificate in the WebSphere truststore. Email Connector version 3.2 and later supports sending emails in plain text, HTML, XML, RTF, and VCF formats. However, Email Connecter does not support sending emails with attachments.

Overview

A “connector” is a Avaya Breeze® platform service that provides connectivity to some application that is external to Avaya Breeze® platform. The Email SMTP Connector communicates with an email server (or more appropriately, a Mail Transfer Agent, or MTA) using SMTP over the well-known SMTP port (port 25).

A service can send a request to the Email SMTP Connector using the Avaya Breeze® platform Email API. The Email Connector queues requests for subsequent delivery (i.e., places the request in its email outbox). The Email Connector will generally send 2 responses to a request: the first response is an acknowledgement that the request was received and queued (placed in the outbox), while the second response indicates the result of the SMTP exchange with the email server.

The Email SMTP Connector requires a small amount of configuration to be operational.

Description

The Email SMTP Connector services its queue (outbox) about every 30 seconds, in what is called a “drain run”. This queuing approach allows the connector to recover from transient email server outages, and also provides a way to throttle outgoing traffic. This traffic throttling allows you to smooth bursts of activity, so that email processing does not compete for resources (e.g., cpu) with other applications that might be higher priority.

At the start of each drain run, the connector checks to see if any requests exist in its outbox. If so, the connector creates a connection pool, if not already created. If the outbox is empty, the connector clears its connection pool. Changes in host configuration will be picked up only once the connection pool has been cleared (i.e., when the outbox is empty). The connections in the connection pool will be maintained as long as the outbox is not empty (i.e., as long as the connector has work to do). The use of a connection pool allows the connector to increase throughput by reusing connections and minimizing the amount of time spent establishing connections.

Configuring the Email connector

About this task

Configure attributes for the Email SMTP Connector using Avaya Aura® System Manager.

Procedure

1. From the Elements panel, select **Avaya Breeze®**.
2. From the **Configuration** menu, select **Attributes**.

3. Select the **Service Globals** tab, then select **EmailConnector** from the **Service** menu.
4. Fill out the Attributes Configuration page for the Email SMTP Connector according to the Email connector field descriptions.

Email connector field descriptions

Field	Description
SMTP Email Host #1 Address	Enter the fully qualified domain name or IP address of a mail transfer agent to which the Email SMTP connector will connect and send email requests. The email connector distributes load between the 2 email hosts. Only 1 email host need be configured.
Concurrent Connections to Host #1	When the Email SMTP connector has work queued up, it will attempt to open this number of concurrent connections to the mail transfer agent.
Host #1 User (optional)	This feature is not currently available.
Host #1 Password (optional)	This feature is not currently available.
SMTP Email Host #2 Address	Enter the fully qualified domain name or IP address of a mail transfer agent to which the Email SMTP connector will connect and send email requests. The email connector distributes load between the 2 email hosts. Only 1 email host need be configured.
Concurrent Connections to Host #2	When the Email SMTP connector has work queued up, it will attempt to open this number of concurrent connections to the mail transfer agent.
Host #2 User (optional)	This feature is not currently available.
Host #2 Password (optional)	This feature is not currently available.
Default Sender's Email Address	Enter an email address that you would like to appear as the email's sender (the sender is the "From" address for an email, but not the "Reply-To" address). If you set this value, all emails sent from the Email SMTP connector will appear to be from this email address, unless overridden in the Email API.
Can service override default sender?	If this value is **true**, then a service can specify a different email sender address (i.e., the email's "From" address) when sending a request using the Email API. If this value is **false**, then the Default Sender's Email Address will always appear to be the sender of the email.
Maximum Number of Emails to Send per Run	This is the maximum number of emails that will be sent on a connection to an email host during the connector's drain run. As an example, if you configure **SMTP Email Host #1 Address** and 5 **Concurrent Connections to Host #1**, no **Host #2 User**, and **Maximum Number of Emails to Send per Run** is set to 2, then a maximum of 10 emails (5 concurrent connections * 2 emails to send per run) will be sent during a drain run.
Maximum Age of an Email Request in Outbox	This is the maximum amount of time a request will sit in the connector's outbox (queue). This condition could be encountered if the connector is unable to connect to an email host, or if the number of emails sent during a drain run is not sufficiently high to empty the outbox in a timely manner.

Field	Description
Maximum Memory Usage for the Outbox	The connector's outbox (queue) is maintained in memory. If you are sending large bursts of emails, you may need to increase this value (maximum is 100MB). If the connector receives a request when the outbox is at its maximum size, an error response will be returned indicating that the outbox is at its size limit.
Test mode enabled?	When you are first testing your service that uses the Email SMTP Connector, you might want to set this to **true** (check the **Override Default** checkbox and change the Effective Value to **true**). When test mode is enabled, the Email SMTP Connector runs through its normal request parsing and validating, and then forms a normal response (but with a status code indicating the connector is in test mode) that is returned to the requesting service.

Using the Email SMTP connector from your service

Use the Email API (which is a part of the larger Avaya Breeze® platform API suite) to request that an email be sent to 1 or more recipients, where a recipient could be in the email's To, Cc, or Bcc list. The Email API is a general API designed to operate with potentially many flavors of Email connector. The API consists of:

- A request object (EmailRequest) that you obtain from a factory and populate with the needed information (e.g., list of recipients, subject, email body). The "send" method on the request sends the request to the Email SMTP Connector. Note that the "send" method is asynchronous, so you need not worry about writing code to spin off a thread to handle a possibly long-running operation.
- A listener object that you implement (to the EmailListener interface) if you are interested in viewing the response from the Email SMTP Connector.
- A response object (EmailResponse) that holds the response from the Email SMTP Connector, provided in the listener.

Following is a snippet from a simple test service. This test service is an HTTP servlet that uses URL query parameters to pass recipients (the to, cc, and bcc parameters), the sender (the from parameter), an email subject (the subject parameter), and an email body (the body parameter). This test servlet also takes an "n" parameter as a count for the number of emails to send, as a way to exercise behavior of the email connector according to various attribute settings.

```
protected void sendRequest(final HttpServletRequest request, final HttpServletResponse response)
        throws IOException
{
        int count = 1;
        final String numEmails = request.getParameter("n");
        if (numEmails != null)
        {
                try
                {
                        count = Integer.valueOf(numEmails).intValue();
                }
                catch (Exception e)
```

```
                }
                        count = 1;

                }
        }
    final PrintWriter w = response.getWriter();
    response.setContentType("text/plain");
    int status = HttpServletResponse.SC_OK;
    for (int i = 0; i < count; i++)
    {
                final EmailRequest email = formRequest(request, i);
                Integer id = EmailTesterRequestCorrelator.INSTANCE.getId();
                final EmailListener listener = new EmailListener(id, i);
                email.setListener(listener);
                EmailTesterRequestCorrelator.INSTANCE.add(id, listener);
                try
                {
                        email.send();
                        w.println("request sent to email connector, id = " + id);
                }
                catch (final Exception e)
                }
                        status = HttpServletResponse.SC_INTERNAL_SERVER_ERROR;
                        w.println("encountered an error: " + e.toString());
                }
        }
        response.setStatus(status);
}

private EmailRequest formRequest(final HttpServletRequest request, int n)
{
        final EmailRequest email = EmailFactory.createEmailRequest();
        final String[] to = request.getParameterValues("to");
        if (to != null)
        {
            email.getTo().addAll(Arrays.asList(to));
        }
        final String[] cc = request.getParameterValues("cc");
        if (cc != null)
        {
            email.getCc().addAll(Arrays.asList(cc));
        }
        final String[] bcc = request.getParameterValues("bcc");
        if (bcc != null)
        {
            email.setFrom(from);
            email.setReplyTo(from);
        }
```

```
            final String subject = request.getParameter("subject");
            if (subject != null)
            {
                  email.setSubject(subject + "[" + n + "]");
            }
            final String body = request.getParameter("body");
            if (body != null)
            {
                  email.setTextBody(body);
            }
return
email; }
```

The bulk of the code in this snippet is HttpServlet code for getting the URL query parameters. The key points are:

- Get an EmailRequest using the factory.final EmailRequest email = EmailFactory.createEmailRequest();
- Set the various attributes of an email, such as: email.setFrom(from), email.setReplyTo(from);, email.setTextBody(body);
- Create a listener and set that listener in the EmailRequest. final EmailListener listener = new EmailListener(id);, email.setListener(listener);
- Send the request to the connector. email.send();

Note that this sample application has a singleton (EmailTesterRequestCorrelator), which tracks requests so that a subsequent query to the application can provide the status of a request.

The listener provides the way to read the response from the email connector. This is simply a class that implements the EmailListener interface and implements the responseReceived method:

```
package com.avaya.zephyr.services.test.emailtester;

import com.avaya.collaboration.email.EmailResponse;
import com.avaya.common.logging.client.Logger;
import com.google.common.base.Joiner;

public final class EmailListener implements com.avaya.collaboration.email.EmailListener
{
    private final Integer id;
    private int status;
    private String detail;
    private String validSent;
    private String validUnsent;
    private String invalid;
    private Logger logger = Logger.getLogger(EmailListener.class);

    public EmailListener(Integer id)
    {
            this.id = id;
```

```
            this.status = 0;
            this.detail = null;
            this.validSent = null;
            this.validUnsent = null;
            this.invalid = null;
    }

    @Override
    public void responseReceived(EmailResponse response)
    {
            status = response.getStatus();
            detail = response.getDetail();
            validSent = Joiner.on(",").join(response.getValidSentAddresses());
            validUnsent = Joiner.on(",").join(response.getValidUnsentAddresses());
            invalid = Joiner.on(",").join(response.getInvalidAddresses());
            logger.fine("ZZZZ responseReceived: " + toString());
    }

    @Override
    public String toString()
    {
            return "id=" + id + ", status=" + status + ", detail=" + detail + ", validSent=[" + validSent + "],
            validUnsent=[" + validUnsent + "], invalid=[" + invalid + "]";
    }
}
```

The response includes a status (to indicate success, failures, etc.), a detail (a string that provides more detail about a failure), and 3 lists indicating the general status of each of the recipients. The “ValidSent” list holds those recipients which the email host accepted as valid (depending on the email host configuration, this might mean, for example, that the email host relayed the email on to another email host). The “ValidUnsent” list holds those recipients which the email host identified as valid, but the email host did not accept the request. The “invalid” recipients list holds those recipients which the email host identified as invalid and rejected. When the email connector returns a status of “partial success”, these 3 lists can help identify those recipients that likely received the email, and those recipients which did not receive an email. Depending on the behavior and/or configuration of the email host, a success indication could actually result in failure or partial success. For example, an email host might accept all recipient addresses and then later send an email reply (i.e., an email sent to the address specified using EmailRequest.setReplyTo) indicating that an address was not reachable.

Chapter 7: Performance and scalability considerations

Performance and scalability considerations

The callIntercepted method of CallListenerAbstract is invoked in a synchronous manner. It is important to consider the impact and minimize or eliminate any synchronous calls in your listener if your listener could start any long-running operation (such as reading from an external database). Only a very limited number of call processing threads can run simultaneously. Under load, the sooner the callIntercepted method returns, the sooner the next call can be processed. To get the best performance, process your calls asynchronously. This means you should create a container-managed asynchronous thread to handle any time consuming work and allow the callIntercepted method to return right away so the next call can be processed. When your thread obtains its desired info, it can use its reference to the original call and allow it to proceed.

The following is an example showing how to do just that. When a call comes into this sample service, it must invoke a REST service to get information about the call. It uses an asynchronous thread to do the REST look up which allows the callIntercepted method to return right away.

Example

Here is the CallListener:

```
@TheCallListener
public class MyCallListener extends CallListenerAbstract
{
	private AsynchronousInvocator asynchronousInvocator;


	// Look up an EJB
	public MyCallListener() throws NamingException
	{
		// Use JNDI to look up the EJB that will do asynchronous operation
		asynchronousInvocator = (AsynchronousInvocator) new
InitialContext().lookup("ejblocal:" +
AsynchronousInvocator.class.getName());
	}


	@Override
	public final void callIntercepted(final Call call)
	{
		call.suspend(); // Always call suspend() before invoking the @Asynchronous method
		asynchronousInvocator.processCall(call); // This bean performs the background
task (e.g.,
REST access to a server)
	}
}
```

Important:

Be sure to invoke the suspend method as shown above (call.suspend()) before invoking the asynchronous method.

Below is the implementation of the stateless session bean, you can see the method processCall() which is annotated with @Asynchronous. In this example, it invokes a REST client method, shouldCallBeAllowed(), which could take some time to complete while it determines if the call should be allowed or dropped. This could just as easily be a database call or any other operation that could block.

Example

```
@Stateless
public class AsynchronousInvocatorImpl implements AsynchronousInvocator
{
    private final RestClient restClient;

    AsynchronousInvocatorImpl(final RestClient restClient)
    {
            this.restClient = restClient;
    }

    @Override
    @Asynchronous
    public void processCall(final Call call)
    {
            try
        {
                    if (restClient.shouldCallBeAllowed())
                    {
                            call.allow();
                    }
                    else
                    {
                            call.drop();
                    }
        }
        catch (final Exception exception)
        {
                    System.err.println("processCall call=" + call);
                    System.err.println("processCall exception=", exception);
        }
    }
}
```

Use container managed threads

Avaya Breeze runtime is based on JEE and JEE best practices must be followed while writing snap-ins.
Using threads that are not managed by the JEE container can lead to:

- Thread exhaustion by uncontrollable creation of threads.
- OutOfMemoryError as each thread consumes memory and huge number of threads can get created.
- Debugging issues with hung threads. No stack traces are shown for hung threads.

The following examples show different ways of using container managed threads in your snap-in.

Example 1:
In a session bean, you can run your code in a container managed thread by annotating a method with "@Asynchronous" as shown below:

```
@Stateless
public class RunAsyncMethodsImpl implements RunAsyncMethod
{
  @Override
  @Asynchronous
  public void asyncMethod()
  {
      //Add your code here that needs to be run asynchronously
     // in a different thread.
   }
}
```

You can invoke this method from a POJO by doing a JNDI lookup of the EJB as shown below.

```
public final class MainClass
{
 private RunAsyncMethod runAsyncMethod;

 public void invokeAsyncMethod()
 {
       runAsyncMethod = (RunAsyncMethod)new  InitialContext().lookup("ejblocal:" +
       RunAsyncMethod.class.getName());
    runAsynchMethod.asynchMethod();
 }
}
```

If you are not writing a session bean, use the container's WorkManager or TimerManager service as shown below.

Example 2:
Submitting a runnable task to an Executor service:

```
    public class MyRunnableTask implements Runnable
```

```
 {
  @Override
  public final void run()
  {
    //Your code to be executed in a
   //separate thread.
  }
 }
```

In the main class do a JNDI look up of “wm/default” to get the container’s ExecutorService and submit the task.

```
    final MyRunnableTask myRunnableTask = new MyRunnableTask();
    final InitialContext initialContext = new InitialContext();
    final ExecutorService executorService = (ExecutorService)    initialContext.lookup("wm/default");
    executorService.submit(myRunnableTask);
```

Example 3:
Scheduling a thread on a regular interval using TimerManager :

```
import commonj.timers.Timer;
import commonj.timers.TimerListener;

public class MyTimerTask implements TimerListener
{
    @Override
    public void timerExpired(Timer arg0)
    {
    //Your code to be executed every regular
    // interval in a separate thread.
    }
}
```

In the main class do a JNDI look up of “tm/default” to get the container’s TimerManager and schedule the task.

```
    final MyTimerTask myTimerTask = new MyTimerTask();
    final InitialContext initialContext = new InitialContext();
        final TimerManager timerManager = (TimerManager)
        initialContext.lookup("tm/default");
        final Timer timer = timerManager.scheduleAtFixedRate(myTimerTask, 60000, 60000);
```

Scaling

Avaya Breeze® platform is designed to run in a clustered configuration. There may be up to 5 Avaya Breeze® platform instances running the same set of services. So, a service should not rely on resources on a specific Avaya Breeze® platform instance such as configuration

files or databases. Instead, these resources should be placed on a remote system that all of the Avaya Breeze® platform instances have access to.

Additional resources

For further information about the Avaya Breeze® platform APIs, check the Avaya Breeze® platform API Javadoc which can be accessed by clicking on a class name in eclipse, pressing F2 and clicking **Open Attached Javadoc in Browser**. If you get stuck, check *Avaya Breeze® platform FAQ and Troubleshooting for Snap-in Developers* and the Avaya Breeze® platform forum on Avaya DevConnect.

Chapter 8: Authorization

Overview

The Authorization Service snap-in is pre-loaded with Avaya Breeze™ Release 3.9 Element Manager. The snap-in provides the following capabilities:

- Authenticate and grant tokens to end users by supporting the OAuth 2.0 Authorization Code Grant flow.
- Authenticate and grant tokens to Avaya Breeze® platform or external authorization clients by supporting the OAuth 2.0 Client Credentials Grant flow.
- Authenticate and grant tokens to end users by supporting the OAuth 2.0 Resource Owner Password Credentials Grant flow.

The following is a brief description of what the three snap-ins, Authorization Service and two test snap-ins, are used for:

- Authorization Service - Is the snap-in that issues the access tokens to the client after successfully authenticating the client itself or the resource owner and obtaining authorization.
- TestAuthorizationClient - Is an authorization client, which according to OAuth2.0 terminology, is an application that makes protected resource requests on behalf of the resource owner and with its authorization.
- TestAuthorizationResource - Is a snap-in or an authorization resource that hosts the protected resources and is capable of accepting and responding to the protected resource requests using access tokens.

Integrating snap-in clients with Authorization Service

About this task

Use the following procedure to enable a snap-in built using the Avaya Breeze® platform SDK to be recognized as an Authorization Client. To understand how the Authorization Clients discover which Authorization Service node or cluster they need to talk to, see “Discovering Authorization Service”.

Procedure

1. Modify the properties.xml file of the client snap-in SVAR, TestAuthorizationClient in the following example, to add the new authorization.client attribute.

```
<attribute name="com.avaya.authorization.client">
    <displayName>Authorization Identifier</displayName>
    <helpInfo>This is used to uniquely identify this snap-in as an Authorization
    Client</helpInfo>
    <validation name="anyString">
        <type>STRING</type>
    </validation>
    <admin_visible>false</admin_visible>
```

```
        <admin_changeable>false</admin_changeable>
        <factory>
                <value>TestAuthorizationClient</value>

                <user_changeable>false</user_changeable>
        </factory
</attribute>
```

Important:

The content that you type within <factory><value> must match the servicename of the snap-in that you have specified during creation of the snap-in. In our example the name specified is TestAuthorizationClient.

2. Use the AuthorizationClientHelper API that is part of the SDK in the com.avaya.collaboration.authorization.client package to get access tokens.

   An example usage of the APIs is provided by the SampleAuthorizationClient class in the com.avaya.collaboration.authorization.sample package. Following is a snippet:

```
AccessToken response = null;
try
{
    response = AuthorizationClientHelper.getAccessToken();
    if (response.getToken() != null)
    {
            return Response.ok(response, MediaType.APPLICATION_JSON).build();
    }
}
```

   The retrieved token is used as a Bearer token in the Authorization header when the client makes a resource request to the resource server. The following APIs are available for getting tokens, the details for which are available in Javadocs:

Oauth 2.0 flow	Entity being authenticated	SDK API used
Authorization Code Grant	End User	AuthorizationClientHelper.getAccessTokenForUser(ServletRequest servletRequest)
Authorization Code Grant	End User	AuthorizationClientHelper.getAuthorizationEndpoint(ServletRequest servletRequest)
Client Credentials	Application	AuthorizationClientHelper.getAccessToken();
Client Credentials	Application	AuthorizationClientHelper.getAccessToken(List<String>scopes)
Resource Owner	End User	AuthorizationClientHelper.getAccessTokenForUser(String userName, String userPassword)
Resource Owner	End User	AuthorizationClientHelper.getAccessTokenForUser(String userName, String userPassword, List<String>scopes)

3. Shut down the helper during snap-in uninstall.

   The API calls mentioned in the previous steps use HTTP connections to talk to the Authorization Service and require an explicit call to shut down when you are uninstalling the snap-in. Therefore you must call the API to clear up system resources. Include the following code snippet wherever you are cleaning up snap-in held resources before an uninstall:

   ```
   AuthorizationClientHelper.shutDown();
   ```

   Once the snap-in is built, loaded and installed on System Manager an entry is created in the Authorization Clients table with the same name you have specified in the authorization.client attribute. In our example the name is TestAuthorizationClient. The Authorization Clients table is located in the **Avaya Breeze® > Configuration > Authorization > Clients** tab.

User Scopes

After successful authentication, the Authorization Service includes scopes of a user in the access token that the service generates.

Scopes from SMGR Profile

If the authenticated user's entry is available in System Manager, the following profile information is included, if configured:

- CommunicationAddress
- MailboxNumber
- StationExtension
- AgentId

Scopes from Oceana Unified Collaboration Administration (UCA)

If the Authorization Service has been configured to query the UCA service to fetch the agent-related information that is provisioned in Avaya Control Manager, after authenticating the user, Authorization Service will make a UCA query to know what role has been configured for the user and will include this information in the access token.

Integrating snap-in resources with Authorization Service

About this task

Use the following procedure to enable a snap-in built using the Avaya Breeze® platform SDK to be recognized as an Authorization Resource.

Procedure

1. Modify the properties.xml file of the resource snap-in SVAR, TestAuthorizationResource in the following example, to add the new resource_server element.

```
<smgr>
	<resource_server>
		<displayName>TestAuthorizationResource</displayName>
		<shortName>ar</shortName>
		<feature>
			<name>write</name>
			<value>standard,privileged,none</value>
		</feature>
		<feature>
			<name>read</name>
			<value>standard,privileged,none</value>
		</feature>
	</resource_server>
</smgr>
```

where,

displayName is the name that will be displayed in the Avaya Breeze® platform Authorization Resource servers page. This must be the serviceName of the snap-in.

shortName is the short version of the displayName. You must choose a globally unique shortName to ensure there is no namespace collision. At runtime, Authorization Service prefixes the shortName specified to each feature while granting an authorization token to a client. For example, read feature in the example code snippet provided above, will be denoted as "featureName" : "ar.read" when a grant is being given to a client. This name is verified by the resource server when a client uses the granted token and requests for a resource.

feature (with a name and one or more values) tags specify the features advertised by an authorization resource. The features specified are available to an administrator to map them to appropriate clients.

2. Handling bearer tokens coming from Authorization Clients.

An Authorization Resource snap-in does not need to talk to the Authorization Service to validate tokens. Bearer tokens contain authorization data signed by the Authorization Service. If the signature validates to true, then the token is valid. The SDK provides an API called AuthorizationResourceHelper, which helps in validating tokens. An example which uses this API is given in the SampleAuthorizationResource class in the package com.avaya.collaboration.authorization.sample in Javadocs. Following is a snippet:

```
String accessToken = bearerToken.substring("Bearer".length()).trim();
try
{
	if (AuthorizationResourceHelper.isAccessTokenValid(accessToken))
	{
		// proceed with logic to serve the request
	}
	else
	{
		// respond back to the client saying the token is invalid
	}
```

```
}
```

The API mentioned above is fine, if you have APIs that really do not need to perform any authorization grant checks on the token. However, more frequently, resource servers have APIs that need to check if the request has sufficient privileges, such as features that had been assigned to clients by an administrator on System Manager. For this case, the AuthorizationResourceHelper API provides a way which retrieves authorization information from the token passed by the client. Following is a snippet:

```
AuthorizationData response =
AuthorizationResourceHelper.getAuthorizationData(accessToken);
List<AuthorizationScope> clientScopes = response.getClientScopeList();

for (AuthorizationScope aScope : clientScopes)
{
if (aScope.getFeatureName().equals("ar.read") &&
aScope.getFeatureValues().contains("standard"))
        {
                // API logic to serve the request
        }
}
```

Apart from client scopes, user scopes are also associated if a token granted to an authenticated user is used to make a resource request. The user's communication profile information is available in the userScopeList. The same is retrieved using the following API call:

```
AuthorizationData response =
AuthorizationResourceHelper.getAuthorizationData(accessToken);
List<AuthorizationScope> userScopes = response.getUserScopeList();
....
```

The token also contains information on the subject retrieved, which is the user's login handle. The following helper API call returns the subject:

```
AuthorizationData response =
AuthorizationResourceHelper.getAuthorizationData(accessToken);
String subject = response.getSubject();
```

Once the snap-in is installed on System Manager an entry is created in the Breeze Resources Servers table with the same name you have specified in the displayNameattribute of the resource_server element. In our example the name is TestAuthorizationResource. The Breeze Resources Servers table is located in the **Avaya Breeze**® > **Configuration** > **Authorization** > **Resource Servers** tab.

Administering grants to an Authorization Client

Before you begin

Install an Authorization Client and an Authorization Resource.

About this task

Use the following procedure to assign grants to an Authorization Client.

Procedure

1. On the System Manager web console, click **Elements** > **Avaya Breeze®**.
2. Click the **Configuration > Authorization > Clients** tab.
3. Select the Authorization Client installed and click **Edit Grants**.

   The Edit Grants page displays grants that have been assigned to the client.

4. To assign a grant to the client, click **New** on the Edit Grants page.
5. On the Create Grants page, do the following:
   a. From the **Resource Name** drop-down list, select the resource name.
   b. From the **Resource Cluster** drop-down list, select the resource cluster.
   c. From the **Feature** drop-down list, select any of the features the resource server advertises.

      Once you select the feature the system displays values associated with that feature in a **Values** table.

   d. Select that appropriate values to be associated with the feature from the **Values** table.
   e. Click **Commit**.

      The Edit Grants page displays the saved grant. This completes the mapping of an Authorization Resource feature to an Authorization Client.

6. On the Edit Grants page, click **Done**.

Discovering Authorization Service

About this task

Use the following procedure to understand how the Authorization Clients discover which Authorization Service node or cluster they need to talk to.

Procedure

1. On the System Manager web console, click **Elements > Avaya Breeze®** .
2. In the navigation pane, click **Cluster Administration**.
3. On the Cluster Administration page, click **New**.
4. On the Cluster Editor page, select the cluster profile of your choice.
5. Specify the Authorization Service node or cluster that the snap-in must integrate with by entering the IP address or FQDN of the node or cluster where the Authorization Service is running in the **Authorization Service Address** field in the Cluster Attributes section.

   An Authorization Client snap-in can also override the cluster value by providing a service attribute in its properties.xml file and updating the attribute under **Avaya Breeze® > Configuration > Attributes** as follows:

```
<attribute name="com.avaya.edp.authorization.service.address">
    <displayName>Authorization Service Address</displayName>
    <helpInfo>FQDN/IP of the node/cluster where Authorization Service is installed</helpInfo>
    <validation name="anyString">
            <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible
     <admin_changeable>true</admin_changeable>
    <factory>
            <value></value>
            <user_changeable>true</user_changeable>
    </factory
</attribute>
```

Authentication mechanisms

Authorization Service supports two mechanisms for authenticating users before authorizing them. The below table provides an overview:

OAuth 2.0 Grant Type (Authorization)	Authentication Mechanism supported
Authorization Code	SAML, LDAP
Resource Owner Password Credentials	LDAP

The following sections describe how to integrate each Grant type with the particular authentication mechanism.

SAML authentication with Authorization Code Grant Flow

Security Assertion Markup Language (SAML) is an XML-based, open-standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider (IdP) and a service provider. For more information, see https://en.wikipedia.org/wiki/ Security_Assertion_Markup_Language_.

The Avaya Breeze® platform Authorization Service acts as an SAML Service Provider when trying to authenticate end-users against an Identity Provider. Authentication is initiated by using an SP-initiated SSO exchange. For more information, see http://docs.oasis-open.org/security/saml/Post2.0/ sstc-saml-tech-overview-2.0.html#5.1.2.SP-Initiated%20SSO:%20%20Redirect/POST%20Bindings %7Coutline_.

In such an exchange, the user attempts to access a resource, for example an (Authorization) Client snap-in (TaskDashboard). Since she doesn't yet have a login session and her identity is maintained by an IdP, she is redirected by the Client snap-in to the Authorization Service, which in-turn redirects her to a configured IdP. After successful authentication with the IdP, she is redirected back to Authorization Service with a SAML assertion. An assertion contains information given by the IdP about the user. For more information, see https://www.oasis-open.org/committees/download.php/ 11785/sstc-saml-exec-overview-2.0-draft-06.pdf_.

The Authorization Service then creates a session for the user and redirects her back to the Client snap-in with an “authorization code”. The Client snap-in then uses the Authorization Service to exchange this code with an access token. The access token is then used to login the user. For more information, see https://tools.ietf.org/html/rfc6749#section-1.3.1 .

Using Servlet Filters to enable SAML redirection

To enable an Authorization Client Snap-in to support SAML, the Breeze SDK provides sample filters which adopters can use to redirect the user to the Authorization Service. Modify the snap-in WAR web.xml to include the filters as shown:

```
<filter>
    <filter-name>AuthorizationCodeFilter</filter-name>
    <filter-class>
            com.avaya.zephyr.services.sample_services.Authorization.AuthorizationCodeFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>AuthorizationCodeFilter</filter-name> <url-pattern>/*
    </url-pattern>
</filter-mapping>

<filter>
    <filter-name>AccessTokenCookieFilter</filter-name>
      <filter-class>

com.avaya.zephyr.services.sample_services.Authorization.AccessTokenCookieFilter
      </filter-class>
</filter>
<filter-mapping>
    <filter-name>AccessTokenCookieFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

AuthorizationCodeFilter

This filter checks if the request contains an authorization code, if not, it redirects the browser to Authorization Service for getting one.

For example, when the user clicks on the home page:

https://<FQDN>/services/TaskDashboard/

The filter will redirect the user to authenticate with the Authorization Service. When the user authenticates successfully, another redirection gives the control back to the AuthorizationCodeFilter, but now with the request containing an authorization code.

AccessTokenCookieFilter

This filter retrieves the authorization code present in the request and uses it to get an access token from the Authorization Service. Once the access token is available, it sets the token as a session cookie and redirects the user back to where she had begun:

https://<FQDN>/services/TaskDashboard/

AccessTokenCookieFilter Service Attributes

The filter uses two service attributes to handle the session cookie creation and user redirection.

SessionCookieName: To provide a name to the session cookie, use the following service attribute in the snap-in properties.xml file:

```
<attribute name="com.avaya.authorization.sessionCookieName">
    <displayName>Session Cookie Name</displayName>
    <helpInfo>AccessTokenCookieFilter uses this attribute to set the user session's cookie
    name.</helpInfo>
    <validation name="anyString">
        <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <admin_changeable>true</admin_changeable>
    <factory>
        <value>dashboard_session</value>
        <user_changeable>true</user_changeable>
    </factory>
</attribute>
```

In the example code above, the filter will create the session cookie with the name “dashboard_session”

ClientRedirectionUri: This attribute is used to redirect the request to another path (for ex, the home page) after setting the cookie session. Modify the properties.xml file to include this attribute:

```
<attribute name="com.avaya.authorization.clientRedirectionPath">
    <displayName>Client Redirection Path</displayName>
    <helpInfo>AccessTokenCookieFilter uses this path attribute to redirect the browser
    to.</helpInfo>
    <validation name="anyString">
        <type>STRING</type>
    </validation>
    <admin_visible>true</admin_visible>
    <admin_changeable>true</admin_changeable>
    <factory>
        <value>/services/TaskDashboard/</value>
        <user_changeable>true</user_changeable>
    </factory>
</attribute>
```

In the example code above, the attribute value has been set to “/services/TaskDashboard/”. This means that the filter, after setting the cookie, redirects the user to:

https://<FQDN>/services/TaskDashboard/ with the session cookie with name “dashboard_session”.

Session Cookie usage to maintain an SAML authentication logged-in session

The cookie, which the filter sets during redirection, has the format:

```
{
    “currentUser”:
    {
        “authdata”:”<access_token>,
        “username”:”testuser”,
        “expiry”:”<unix_time>”
    }
}
```

The UI to be rendered can check for the presence of this cookie when the user tries to access the page. If the cookie is present, the access token can be retrieved and set as a global Authorization header so that requests to resources are authorized:

```
$rootScope.globals = $cookies.getObject('dashboard_session') || {};

if($rootScope.globals.currentUser) {
    $http.defaults.headers.common['Authorization'] = 'Bearer ' +
    $rootScope.globals.currentUser.authdata;
}
```

Refer to the Authorization Sample snap-ins in the Breeze SDK for a detailed walkthrough of the flow and sample code.

LDAP authentication with Authorization Code Grant flow

Similar to SAML, the Authorization Service supports LDAP authentication for authenticating and authorizing users with the Authorization Code Grant flow.

Consider again the example of a user trying to access a resource using the (Authorization) Client snap-in (TaskDashboard). Since she doesn’t yet have a login session and her identity is validated against an LDAP server, she is redirected by the Client snap-in to the Authorization Service, which presents her with a login screen.

After successful authentication, the Authorization Service then creates a session for the user and redirects her back to the Client snap-in with an authorization code. The Client snap-in then uses the Authorization Service to exchange this code with an access token. The access token is then used to login the user.

Using Servlet Filters to enable LDAP redirection

Please refer to the SAML section above on the usage of filters to enable the Client snap-in to support this flow. The same filter usage applies for LDAP authentication.

Session Cookie usage to maintain an LDAP authentication logged-in session

Please refer to the SAML section above on how a session cookie set by the filters can be used to maintain a logged-in user session. The cookie creation/usage remains the same for LDAP authentication.

Choosing SAML or LDAP in a deployment

To choose an Authentication Mechanism, go to: System Manager> Avaya Breeze® >

Configuration > Authorization > Authentication Mechanism

SAML authentication

SAML deployments require agreements between system entities regarding identifiers, binding support and endpoints, certificates and keys, and so forth. A metadata specification is useful for describing this information in a standardized way.

The system entities involved here are the Avaya Breeze® platform Authorization Service (acts as a Service Provider (SP)) and a far-end IdP. The metadata of both the entities are XML files which need to be exchanged between the them:

- The SP Metadata is used for configuring the SP at the IdP.
- The IdP Metadata is used for configuring the IdP at the SP.

Getting the SP (Authorization Service) Metadata

The Authorization Service, after installing in an Avaya Breeze® platform cluster, generates its metadata on a per-node basis.

The metadata file is available for download by accessing the following path on each node:

https://<SecurityModuleIP>:9443/services/AuthorizationService/spmetadata

Configuring the SP (Authorization Service) at the IdP

The downloaded metadata file needs to be used while configuring the Authorization Service as a Service Provider at a far-end IdP. Please refer to the Authorization Section in *Administering Avaya Breeze® platform* for an example on how to configure the Authorization Service SP as a Relying Party on the Active Directory Federation Services.

Configuring the IDP on SMGR

The Authorization Service acting as a Service Provider, picks the IdP details from the SAML Authentication Mechanism configuration.

To configure an IdP, choose Authentication Mechanism > SAML. For details on the configuration steps, refer to *Administering Avaya Breeze® platform*.

Enabling SAML Profile for Authorization

When the Authorization Service is installed, the SAML component is not enabled/started by default. After configuring the IDP metadata, the profile needs to be enabled. This can be done by changing the following service attribute:

Go to Avaya Breeze® > Configuration > Attributes > Service Clusters

Choose the Cluster where Authorization Service has been installed and choose Service as Authorization Service. On the attribute named “SAML Profile”, change the value to choose “Deploy” and commit the changes.

LDAP authentication

LDAP Authentication is enabled by default. For information on how to configure LDAP and synchronize the server with System Manager, see:

- *Administering Avaya Breeze® platform*
- *Avaya Aura® System Manager 7.0 LDAP Directory Synchronization Whitepaper*

LDAP authentication with Resource Owner Password Credentials Grant Type

Use of Resource Owner Password Credentials flow is discouraged unless the Authorization Code Grant flow is not possible, e.g. due to lack of a web user interface on the Client.

Consider again the example of a user trying to access a resource using the (Authorization) Client snap-in (TaskDashboard). Since she doesn't yet have a login session and her identity is validated against an LDAP server, the TaskDashboard snap-in provides her with a Login page to enter her credentials. The Avaya Breeze® platform Authorization SDK provides the following APIs to authenticate the credentials entered on the form:

Function	SDK API
Authenticate User	AuthorizationClientHelper.getAccessTokenForUser(String userName, String userPassword)
Authenticate User with Scopes	AuthorizationClientHelper.getAccessTokenForUser(String userName, String userPassword, List<String>scopes)

The difference between the two APIs is that the second one asks for an access token with Authorization capabilities including a specific list of scopes. The first API will get an access token for all the scopes a user/client has been assigned with.

System Manager datasource synchronization caveats for LDAP configuration

To know which LDAP server to talk to Authorization Service uses the datasources configured in System Manager. You can configure two types of datasources in System Manager:

- Active Directory
- Non-Active Directory (includes OpenLDAP)

Authorization Service requires the username parameter provided in the APIs mentioned in the "LDAP integration for authenticating users" section to be in the user@domain format. This is because Authorization uses the user@domain format to fetch the relevant communication profile data of the respective user and include it in the token being generated for the user.

Synchronization is not necessary if a single Active Directory datasource is provisioned

Active directory understands usernames in the user@domain format and therefore, it is not necessary to synchronize Active Directory users in System Manager. Authorization Service directly uses the username parameter that is sent to the API to authenticate the user against the configured LDAP server. However you must configure the LDAP server under System Manager Directory Synchronization.

Synchronization is mandatory if multiple datasources are provisioned

If multiple datasources are provisioned in System Manager, then synchronization of users from all the datasources is mandatory. In this case, Authorization Service always tries to fetch the Distinguished Name (DN) of the provided username from System Manager and use the DN to authenticate the user.

Synchronization is mandatory if a Non-Active Directory (OpenLDAP) datasource is provisioned

OpenLDAP requires LDAP bind requests to contain the Fully Qualified Distinguished Name (FQDN), for example: uid=testuser,ou=people,dc=avaya,dc=com, when authenticating users.

However, you cannot expect users to type in the DN when they are logging in, which means that the userName parameter in the APIs mentioned in the “LDAP integration for authenticating users” section cannot be passed with a DN. Therefore, Authorization Service requires OpenLDAP server deployments to enforce a mandatory synchronization of users with System Manager when such a datasource is being provisioned. When this is done, the user can provide the userName in the user@domain format, but Authorization Service retrieves the DN of the synchronized user mapped to the username available from System Manager and uses the DN while authenticating the user against OpenLDAP.

Chapter 9: Service monitors

Health Monitoring Service

The Health Monitoring Service monitors the status and health of various platform components. Several of the monitored components are only for use by Avaya snap-ins. The Cluster Database is the component that is monitored by Health Monitoring Service and is available for general use.

If your snap-ins uses the Cluster Database, you can declare it as a dependency in properties.xml. You can also declare actions to be taken when the Cluster Database becomes unavailable.

Snap-in dependencies and actions

You snap-in can declare platform dependencies and actions in properties.xml. A sample declaration is provided here:

```
<dependencies>
        <dependency type="platform">
                <name>CLUSTER_DB</name>
                <actions>
                        <install-time-action>ADMIN_WARNING</install-time-action>
                        <run-time-action>STOP</run-time-action>
                        <startup-time-action>DONT_START</startup-time-action>
                </actions>
        </dependency>
</dependencies>
```

Actions

Action	Type	Description
Install time actions	Admin Warning	Element Manager displays a warning that a dependent platform component is unavailable. Administrators can choose to continue or cancel the installation.
Run time actions	Stop	Health Monitoring Service stops the snap-in. The system will start the snap-in when the dependent platform components are available. Administrators can manually start snap-ins using the Start button on the Service Status page of an Avaya Breeze® platform instance.
Startup time actions	Don't Start	The snap-in cannot be started if the dependent components are not available.

Measuring snap-in resource usage

Before you begin

Make sure your snap-in is not installed in the cluster.

Procedure

1. Reboot all nodes in the cluster.
2. Reset the peak usage for the cluster in which you plan to test.
3. Get a resource usage baseline:
   - If your snap-in is a call intercept type, then run traffic or load at the level you desire to measure against the cluster without your snap-in.
   - If not, then let the system idle with no load.
   - There are many platform operations that occur in the background, so you should run this for at least two hours.
4. Record the peak value of all resources on all nodes in the cluster.
5. Install your snap-in.
6. Reset the peak usage again.
7. Run traffic or load at the level you desire to measure against the cluster with your snap-in.
   - Again, do this for at least two hours.
   - If your snap-in runs some of its own operations in the background, then run for as long as it takes to execute all of these operations.
8. Record the peak value of all resources on all nodes in the cluster.

Result

The difference between the two peaks is the incremental resource usage for your snap-in.