Avaya Breeze™ Snap-in Development Guide

Release 3.1
Issue 3
May 2016

REVISED: March 24, 2016

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 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.

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 or to 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 “Open Source Software” or “OSS” is as defined by the Open Source Initiative (“OSI”) and is software licensed under an OSI approved license as set forth at http://www.opensource.org/docs/osd (or such successor site as designated by OSI).

1.9 “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.

1.10 “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.

1.11 “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.

1.12 “Sample Application Code” means Software provided for the purposes of demonstrating functionality of an Avaya product through the Avaya Software Development Kit.

1.13 “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 the Sample Application Code with Permitted Modifications, provided that such distribution is subject to an end user license agreement 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.

C. Except as expressly authorized by this Agreement, and unless otherwise permitted by the applicable law, Licensee acknowledges and agrees that the foregoing license does not include any right to distribute, license, translate, publish, or display the SDK, Specification Documents or Documentation or any copy or part thereof. Licensee represents and warrants that it will not use, modify, or distribute the redistributable Client Libraries in any manner that causes any portion of the redistributable Client Libraries that is not already subject to an OSS license to become subject to the terms of any OSS license.

D. 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).

E. 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.

F. 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.

2.2 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, trade marks 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: https://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.

2.6 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.

2.7 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 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.

2.13 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 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, black boxes, malware, trapdoors, and other mechanisms to allow remote/hidden attacks or access through unauthorized computerized command and control, and will not contain any other computer software routines designed to spy, monitor traffic (network sniffers, keyloggers), 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, 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.

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, worldwide license under any and all of Licensee's Intellectual Property rights related to any Permitted Modifications, to use, employ, practice, make, have made, sell, and/or otherwise exploit any and all Permitted Modifications.

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 derivative 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 derivative 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 return to Avaya all Confidential Information (including copies thereof) in Licensee's possession, custody, or control upon termination of this Agreement at any time and for any reason. 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 AND WILLFUL MISCONDUCT, 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 AND WILLFUL MISCONDUCT, IN NO EVENT SHALL AVAYA'S TOTAL LIABILITY TO LICENSEE IN CONNECTION WITH, ARISING OUT OF OR RELATING TO THIS AGREEMENT EXCEED FIVE HUNDRED DOLLARS ($500). THE PARTIES AGREE THAT THE LIMITATIONS SPECIFIED IN THIS SECTION WILL APPLY EVEN IF ANY LIMITED REMEDY PROVIDED IN THIS AGREEMENT IS FOUND TO HAVE FAILED OF ITS ESSENTIAL PURPOSE.

9.0 INDEMNIFICATION.

Licensee shall indemnify and hold harmless Avaya, Affiliates and their respective officers, directors, agents, suppliers, customers and employees from and against all claims, damages, losses, liabilities, costs, expenses, and fees (including fees of attorneys and other professionals) 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, including, but not limited to, products liability claims and claims of infringement of third party Intellectual Property rights.

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 of this Agreement, Licensee will immediately cease using the SDK, and Licensee agrees to destroy all adaptations or copies of the SDK and Documentation, or return them to Avaya upon termination of this License.

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

11.0 ASSIGNMENT.

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

12.0 COMPLIANCE WITH LAWS.

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

13.0 WAIVER.

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

14.0 SEVERABILITY.

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

15.0 GOVERNING LAW AND DISPUTE RESOLUTION.

This Agreement and any dispute, claim or controversy arising out of or relating to this Agreement (“Dispute”), including without limitation those relating to 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.

Any Dispute shall be resolved in accordance with the following provisions. The disputing party shall give the other party written notice of the Dispute. The parties will attempt in good faith to resolve each Dispute within thirty (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. 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 these procedures and within these timeframes, 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 any or all other parties 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 this Agreement 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 each party will 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 shall 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.

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 above, 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 above with regard to arbitration of Disputes that arise anywhere other than in the United States or are based upon an alleged breach committed anywhere other than in the United States, each party to this Agreement consents to the exclusive jurisdiction of those courts, including their appellate courts, for the purpose of all actions and proceedings.

The parties agree that the arbitration provision in this section 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. Nothing in this section 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. In addition and notwithstanding the foregoing, Avaya shall be entitled to take any necessary legal action at any time, including without limitation seeking immediate injunctive relief from a court of competent jurisdiction, in order to protect Avaya's intellectual property and its confidential or proprietary information (including but not limited to trade secrets).

16.0 IMPORT/EXPORT CONTROL.

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

17.0 AGREEMENT IN ENGLISH.

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

18.0 ENTIRE AGREEMENT.

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

19. REDISTRIBUTABLE CLIENT FILES.

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

Contents

Chapter 1: Creating a service project

Introduction

This guide explains the key concepts needed to develop a collaboration service to run on the Avaya Breeze™. This guide assumes that you have the Avaya Breeze™ SDK installed in your environment. If you do not yet have the SDK, refer to *Getting Started with the Avaya Breeze™ SDK*.

You must also have JDK 1.7 installed. Only Java version 1.7 is supported.

Project skeletons

The recommended procedure for creating a collaboration 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™ 3.0 Maven Archetype, remove the `.m2/repository` directory 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™ 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™ and System Manager.
- serviceVersion: A version string used in Avaya Breeze™ 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

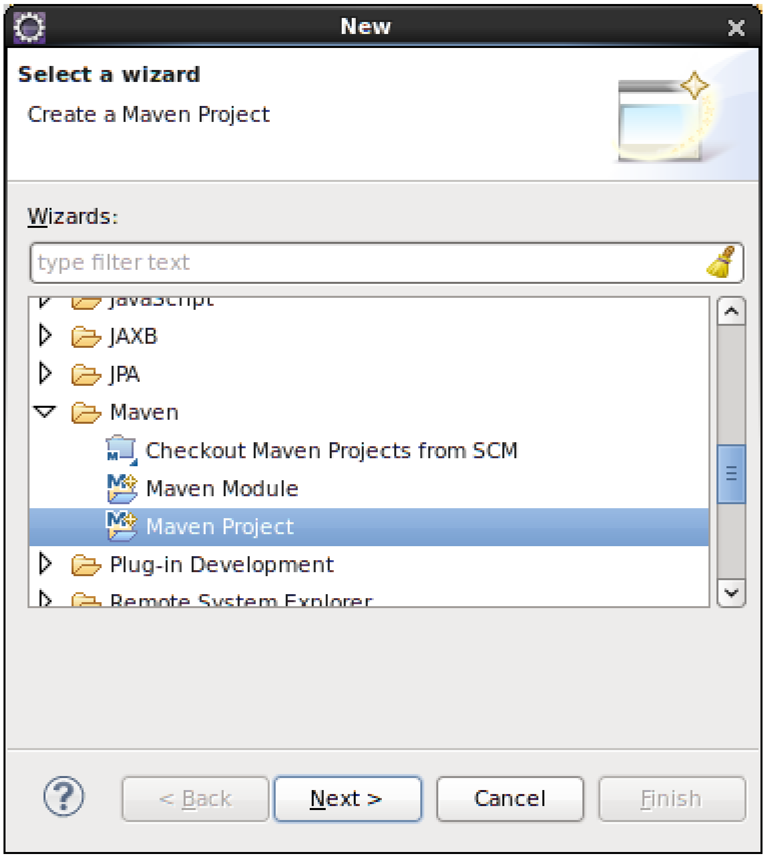
Example Avaya Breeze™ values

- serviceName: TestService
- serviceVersion: 1.0.0.0.0

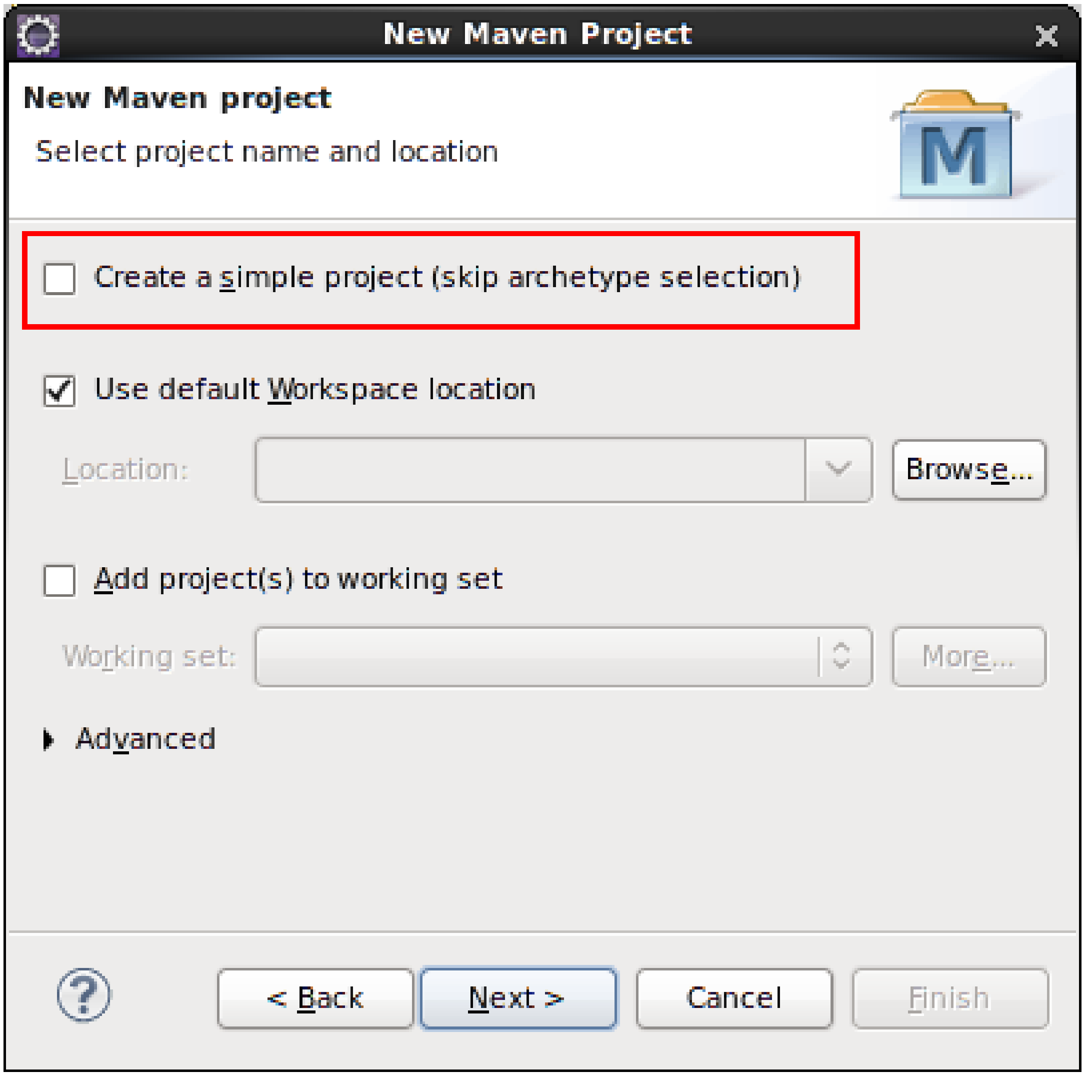
Creating a project skeleton using Eclipse

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**.

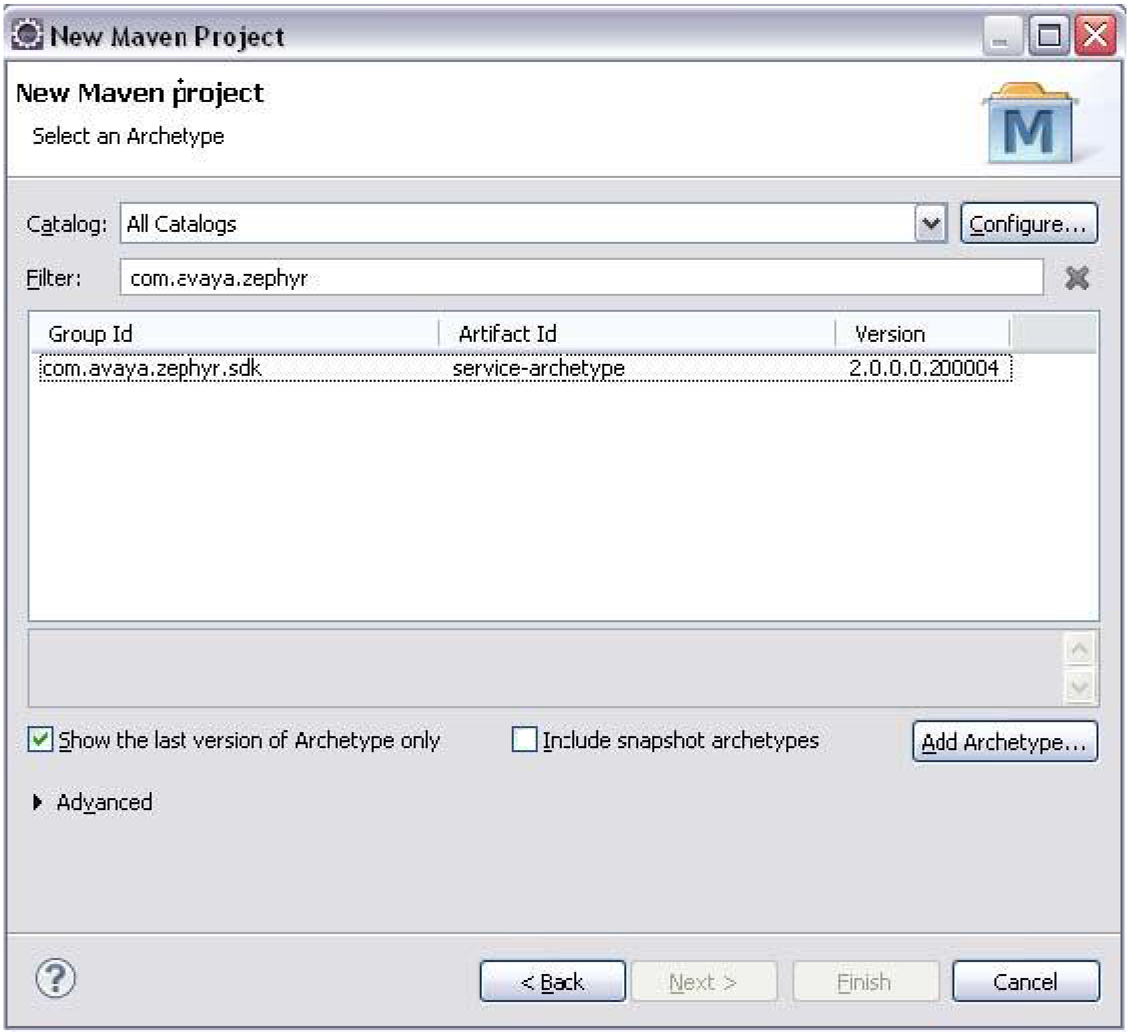


3. Clear the **Create a simple project** check box, and click **Next**.

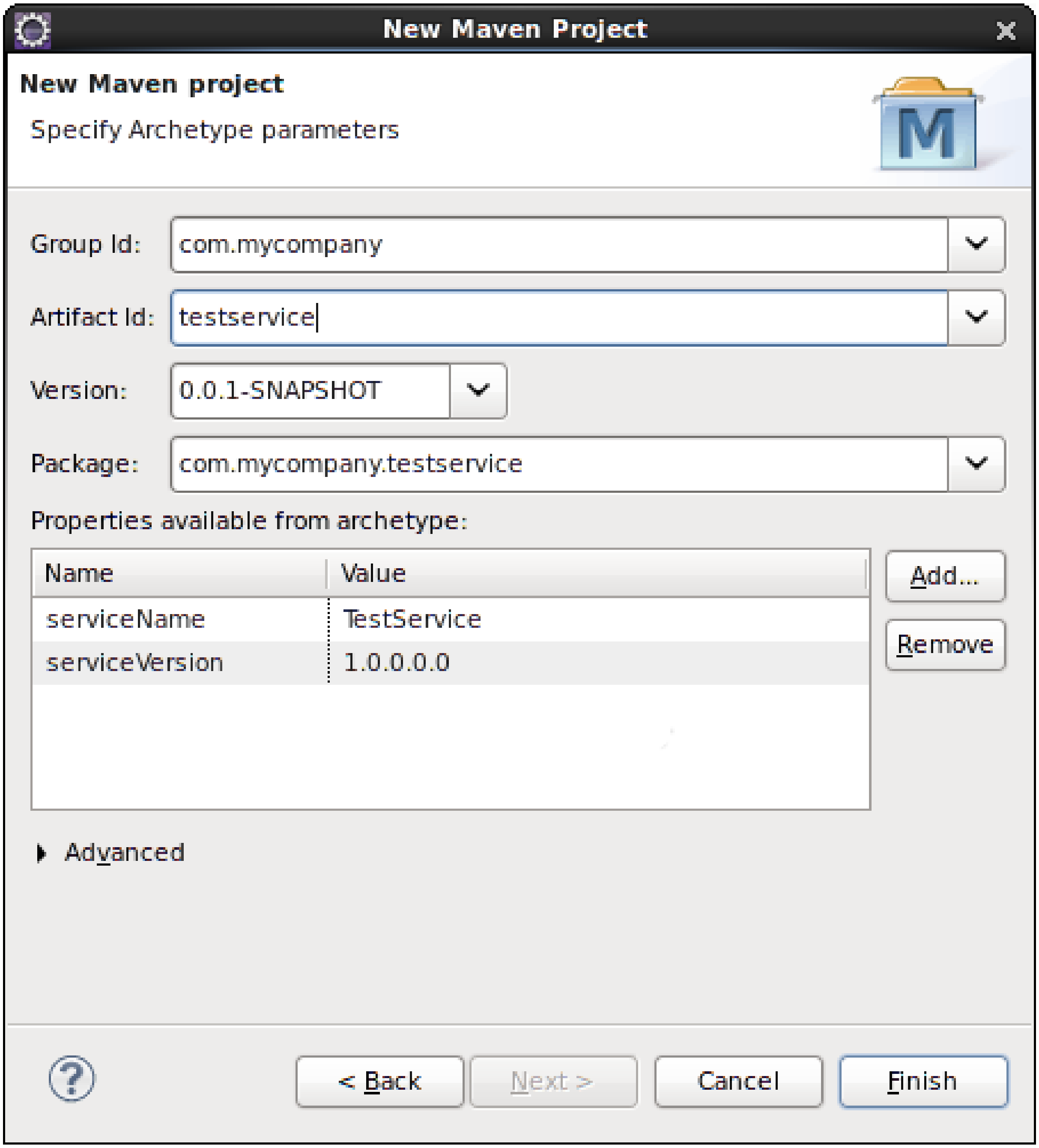


4. Enter `com.avaya.zephyr.sdk` in the **Filter** field on the archetype selection screen.

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™ 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.

```
$mvn archetype:generate -DarchetypeGroupId=com.avaya.zephyr.sdk -DarchetypeArtifactId
=service-archetype
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] Archetype [com.avaya.zephyr.sdk:service-archetype:1.0.0.0.0-SNAPSHOT] found in
 catalog local
Define value for property 'groupId': : com.mycompany
Define value for property 'artifactId': : testService
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  com.mycompany: : com.mycompany.testService
[INFO] Using property: serviceName = TestService
[INFO] Using property: serviceVersion = 1.0.0.0.0
Confirm properties configuration:
groupId: com.mycompany
artifactId: testService
version: 1.0-SNAPSHOT
package: com.mycompany.testService
serviceName: TestService
serviceVersion: 1.0.0.0.0
 Y: : Y
```

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™-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 3 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 3 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.

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” , “servlet-mapping” 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>`.

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();
            }
}
```

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 addtion to diverTo/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.

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

Life Cycle Management

The ServiceLifeCycle interface of the Avaya Breeze™ SDK defines service initialization and destroy 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");
        }
     }
}
```

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

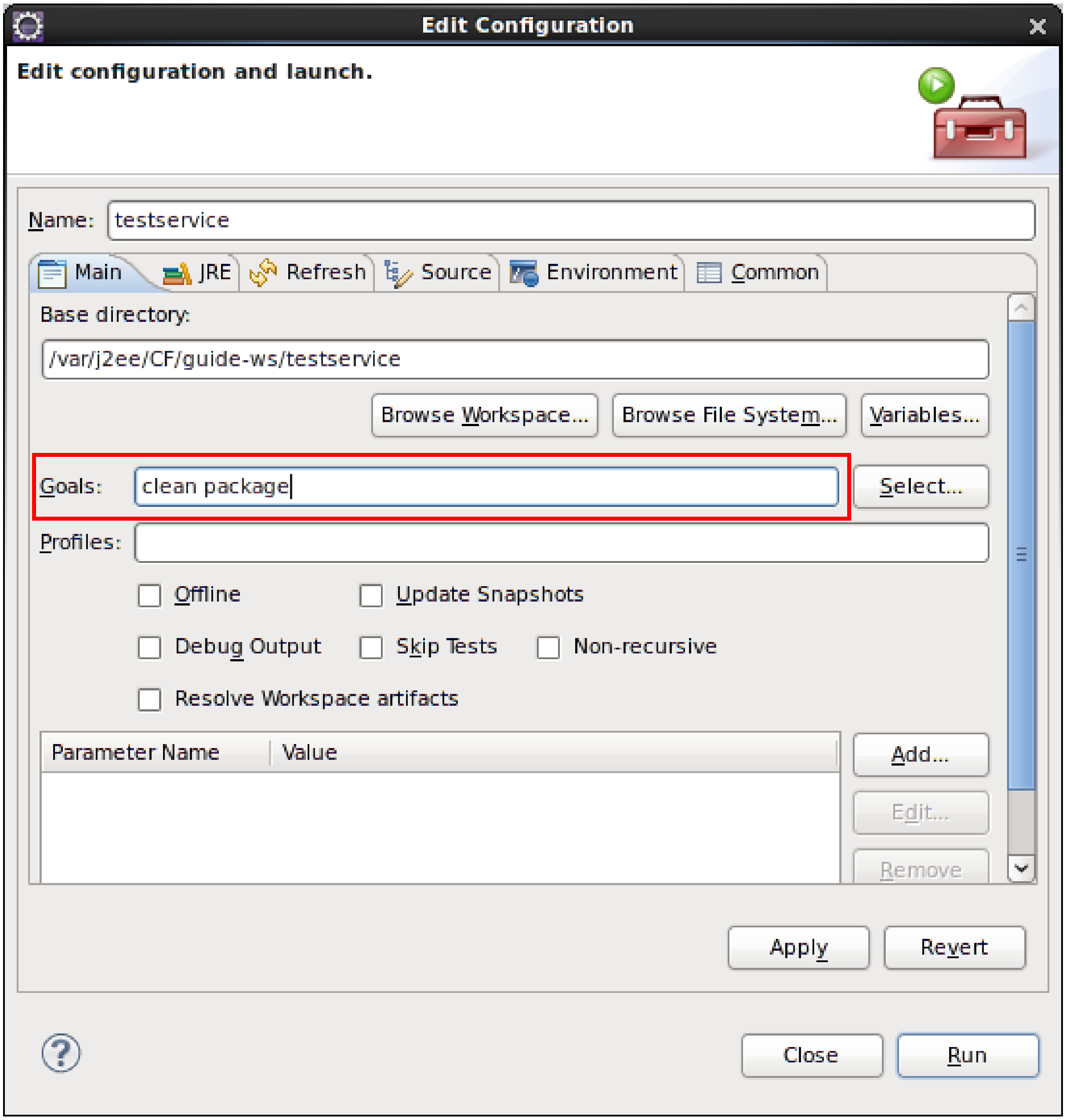
1. Right click on the parent project (testService in the example above) in the project explorer and click **Run as** > **Maven build...**.
2. In the window that appears, enter `clean package` in the **Goals** field and select **Run**.

   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.


Edit Configuration
Edit configuration and launch.
Name: testservice
Main
JRE
Refresh
Source
Environment
Common
Base directory:
/var/j2ee/CF/guide-ws/testservice
Browse Workspace...
Browse File System...
Variables...
Goals: clean package
Select...
Profiles:
Offline
Update Snapshots
Debug Output
Skip Tests
Non-recursive
Resolve Workspace artifacts
Parameter Name
Value
Add...
Edit...
Remove
Apply
Revert
Close
Run

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.

Third Party Jars

Many Third Party Jars are used by the platform and are available to use by snap-in developers. This section lists those jars along with the version. To insure compatiblity, these are the only versions of these jars that should be used.

Jars in lib/ext

- activation-1.0.2.jar
- aopalliance-1.0.jar
- apache-cassandra-2.0.10.jar
- apache-cassandra-clientutil-2.0.10.jar
- archive-jaxb-3.1.0.0.44013.jar
- asm-common.jar
- AsmEventCodes.jar
- avaya-commons-io-3.0.jar
- avaya-commons-lang-3.0.jar
- avaya_logging_client.jar
- AVAYA-SPI.jar
- CAR-3.1.0.0.44013.jar
- car-xsd-3.1.0.0.44013.jar

- cassandra-driver-core-2.0.0.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-lang3-3.1.jar
- commons-pool-1.5.4.jar
- concurrentlinkedhashmap-lru-1.3.2.jar
- dom4j-1.6.1.jar
- gson-2.2.2.jar
- guava-18.0.jar
- hamcrest-all-1.3.jar
- jackson-annotations-2.1.4.jar
- jackson-core-2.1.4.jar
- jackson-core-asl-1.9.2.jar
- jackson-databind-2.1.4.jar
- jackson-mapper-asl-1.9.2.jar
- java-facade-3.1.0.0.44013.jar
- javassist-3.16.1-GA.jar
- jboss-trust-logging-7.0.0.7-14565-SDK-1.0.jar
- jni-libs.jar
- libthrift-0.7.0.jar
- log4j-rolling-appender-20100605-1200-1.2.9.jar
- metrics-core-2.2.0.jar
- netty-3.6.6.Final.jar
- postgresql-9.2-1003.jdbc4.jar
- reflections-0.9.9-RC1.jar
- sip-common.jar
- snappy-java-1.0.5.jar
- spring-aop-4.1.5.RELEASE.jar
- spring-beans-4.1.5.RELEASE.jar
- spring-context-4.1.5.RELEASE.jar
- spring-core-4.1.5.RELEASE.jar

- spring-expression-4.1.5.RELEASE.jar
- spring-jdbc-4.1.5.RELEASE.jar
- spring-tx-4.1.5.RELEASE.jar
- srAgentProxy-3.1.0.0.44013.jar
- tmclient-7.0.0.7-14565-SDK-1.0.jar
- uca-notification-dao-1.45.jar
- uca-notification-util-1.45.jar
- xml-apis-1.0.b2.jar
- xmlpull-1.1.3.1.jar
- xpp3_min-1.1.4c.jar
- xstream-1.4.4.jar
- zephyrDataAPI-3.1.0.0.44013.jar
- zephyrDataApiFactory-3.1.0.0.44013.jar
- zephyrDM-3.1.0.0.44013.jar
- zephyrEncryptDecrypt-3.1.0.0.44013.jar
- zephyrUtilities-3.1.0.0.44013.jar

JARs in lib/ce_shared

- amclientsdk-9.5.4_RTM.jar
- mscontrol-1.0.jar
- smc-api-3.1.0.0.0.jar
- collaborationBusAPI-3.1.0.0.0.jar
- mscontrol-impl-ams-8.1.0.0.44003.jar
- smc-api-impl-3.1.0.0.0.jar
- collaborationBusCore-3.1.0.0.0.jar
- openssoclientsdk-9.5.4_RTM.jar
- smc-call-reconstruction-3.1.0.0.0.jar
- collaborationBusFactory-3.1.0.0.0.jar
- opensso-sharedlib-9.5.4_RTM.jar
- sms-api-3.1.0.0.0.jar
- collaborationBusListener-3.1.0.0.0.jar
- pfaServices-3.1.0.0.0.jar
- sms-api-impl-3.1.0.0.0.jar
- commons-logging-1.1.3.jar
- ports-api-3.1.0.0.0.jar
- speech-search-query-8.1.0.0.44003.jar

- datagrid-api-3.1.0.0.jar
- ports-api-impl-3.1.0.0.jar
- ssal-common-8.1.0.0.44003.jar
- eac-3.1.0.0.jar
- schedConf-api-3.1.0.0.jar
- ssal-well-behaved-8.1.0.0.44003.jar
- email-api-3.1.0.0.jar
- schedConf-api-impl-3.1.0.0.jar
- ssl_util-api-3.1.0.0.jar
- email-api-impl-3.1.0.0.jar
- schemas-3.1.0.0.jar
- ssl_util-api-impl-3.1.0.0.jar
- eventing-api-3.1.0.0.jar
- serviceActivity-api-3.1.0.0.jar
- zephyrDataUtil-3.1.0.0.jar
- eventing-api-impl-3.1.0.0.jar
- serviceActivity-api-impl-3.1.0.0.jar
- zephyrHelper-3.1.0.0.jar

Next steps

Congratulations! You've 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 the document *Quick Start to Deploying Avaya Breeze™ Snap-ins*. 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 2: Developing the service

Developer update method

Avaya Breeze™ is designed to install services from a central server, the System Manager, across the network to multiple Avaya Breeze™ servers. Consequently, movement of a service to a Avaya Breeze™ 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™ 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 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™ 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™ 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™. 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 probably already thinking about the improvements you want to make for the next version. Avaya Breeze™ 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™ 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 that 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 greater 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 can't 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's 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.

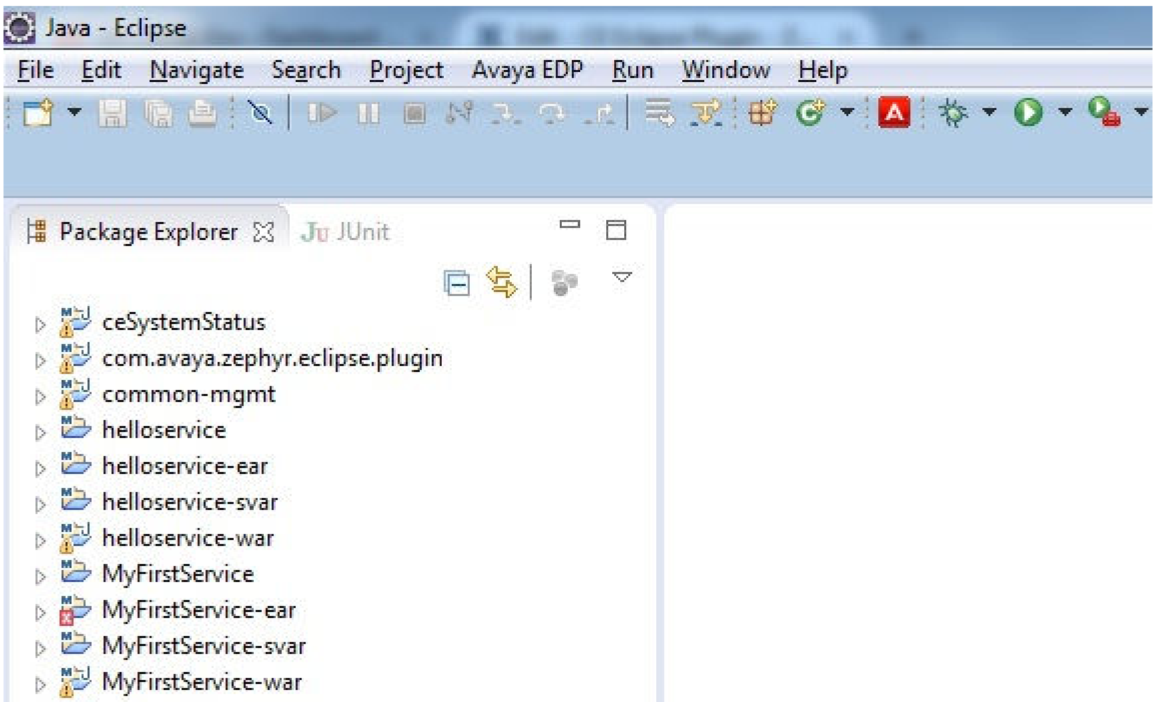
- If "preferred" has been specified, then the designated "preferred" version of the service will be invoked for users with the given service profile. This is similar to the specific version setting, but has the added benefit that the preferred version is applicable to both calls and HTTP messages. If 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*™ guide explains how to assign services to service profiles and how to designate a service version as being the preferred version.

The eclipse plug in

Before you begin

Locate the plug-in icon.



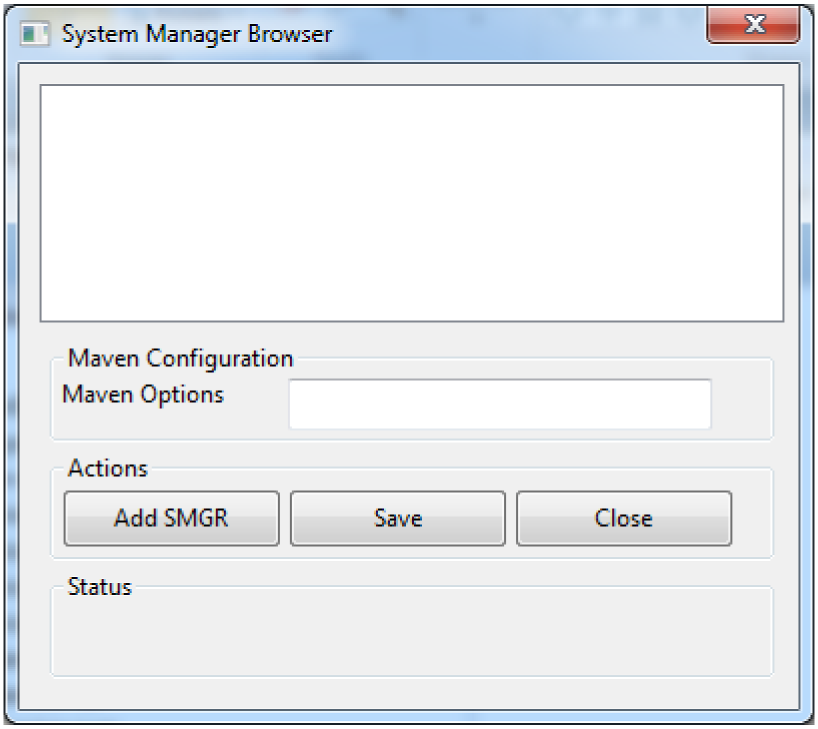
About this task

The eclipse plugin allows the developer to perform actions on the System Manager and Avaya Breeze™ servers right from eclipse. It comes with the Avaya Breeze™ SDK. The Avaya icon will be visible on the tool bar to launch the System Manager configuration window.

Procedure

1. Click the Avaya icon.

   The system displays the System Manager Browser pop-up window as seen in the following screenshot.

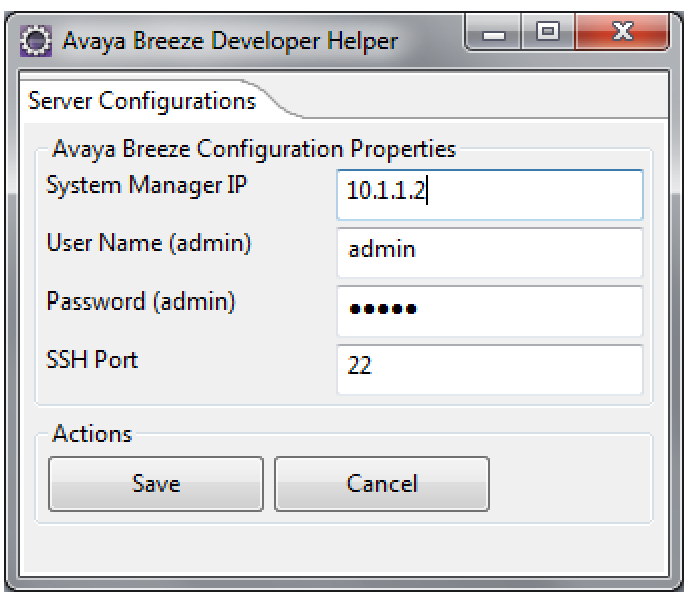


2. Right-click on the white area shown in the System Manager Browser popup to display an add System Manager button.

   The system displays the System Manager Configuration window.

3. Enter the **System Manager IP** and login credentials for an admin user and click **Save**.

   This should fetch all the cluster information from the System Manager and display a tree structure of all the clusters and Avaya Breeze™ nodes along with default populated Avaya Breeze™ configuration details.

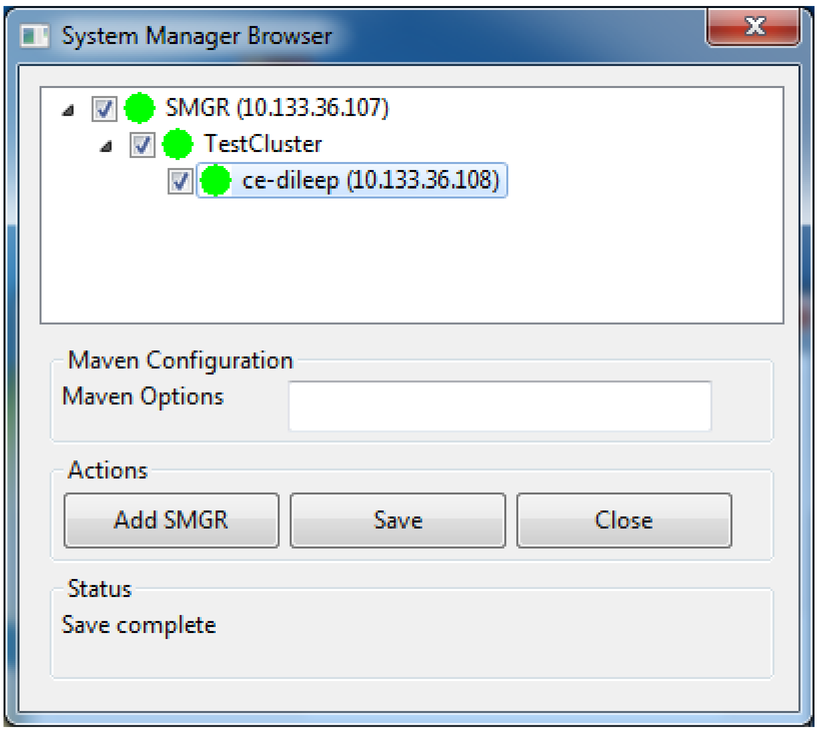


A green status circle for the connectivity status indicates the System Manager and Avaya Breeze™ servers in a cluster are reachable and have the correct credentials.

A red status circle indicated either system is not reachable or provided credentials are incorrect.

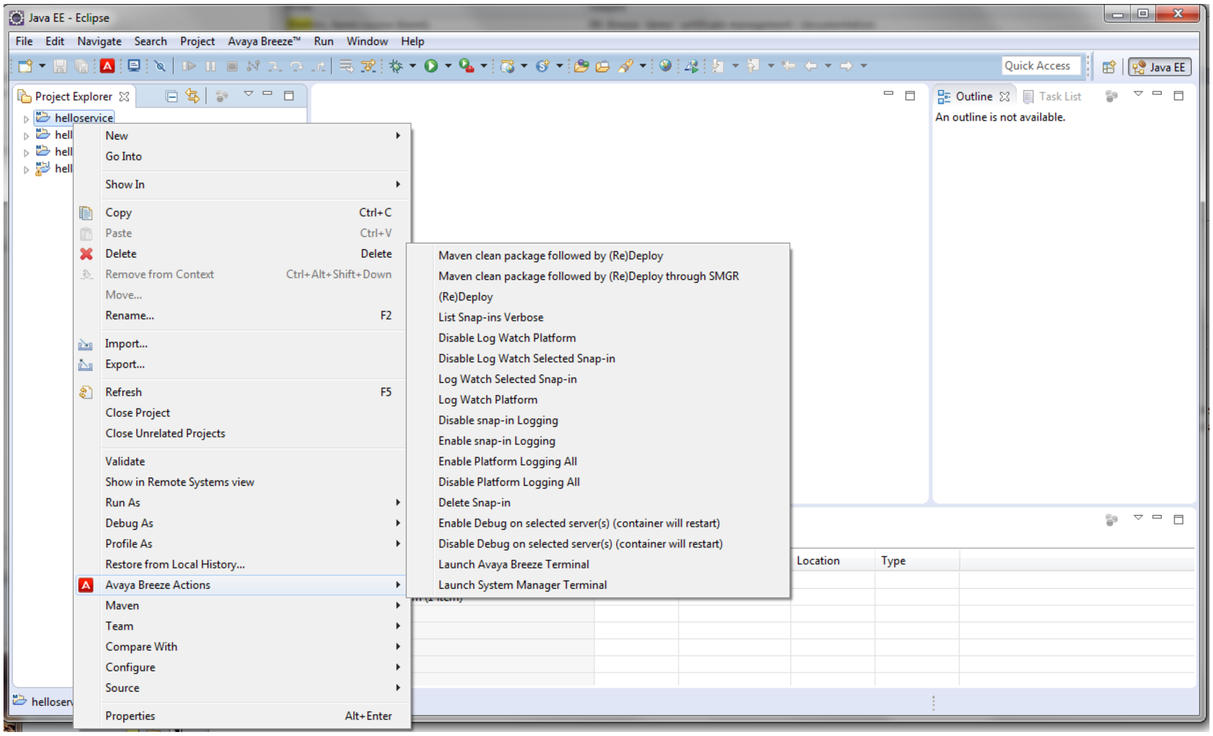
4. To change the credentials of System (Avaya Breeze™/System Manager), select the system and right click to display the **Edit System** option.
5. Click the **Edit System** option to display a popup server configuration window for credentials update.
6. Provide correct credentials and click **Save**.

   The connectivity status will display as green if eclipse plug-in can connect with new credentials.



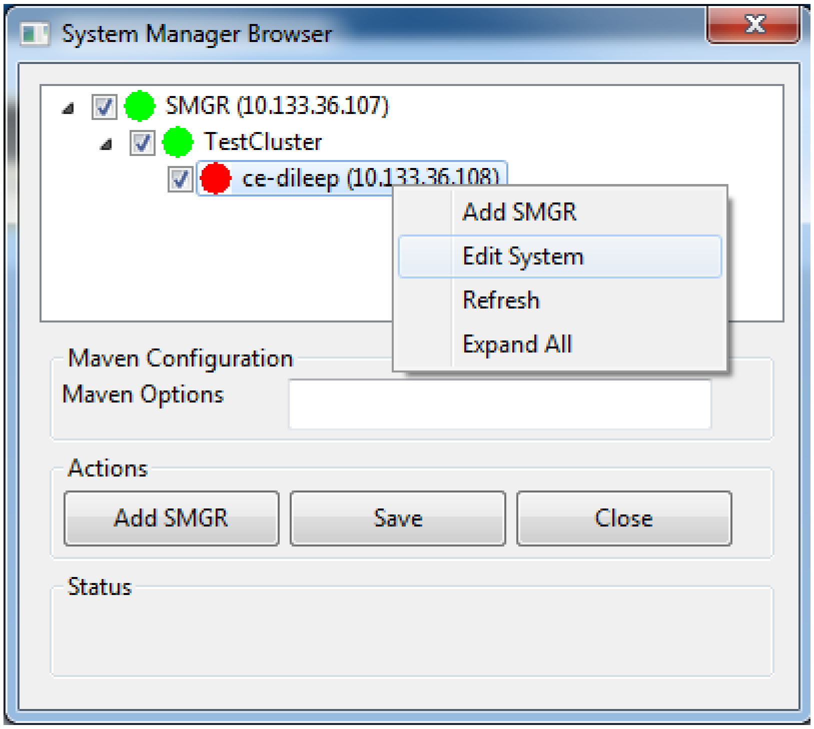
7. Repeat until all the Avaya Breeze™ servers being used are shown as green.
8. Check the System Manager, cluster and Avaya Breeze™ that the plugin will use.

9. Now the plugin is configured, here is how to use it. Right click on the target Avaya Breeze™ project to get the pull down menu containing the **Avaya Breeze™ Actions** menu.



Manage SMGR and Avaya Breeze™ nodes

The following administratative actions are supported for nodes (i.e., SMGRs or Avaya Breeze™s) in System Manager Browser.



Add SMGR	The action will help to add SMGR in System Manager Browser.
Edit System	The SMGR/Avaya Breeze™ node will be edited by this action. Click on **Edit System** and a popup server configuration window for credentials update will appear (shown procedure step 3). Provide correct credentials and click **Save**.
Refresh	Any changes made on SMGR/Avaya Breeze™ will be reflected when click on action Refresh. Previous selection will be lost and plugin developer needs to select cluster and Avaya Breeze™ again and click on **Save for further operations.**
Expand All	This action will expand tree nodes (SMGRs /Avaya Breeze™).

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™ 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™. 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™ 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™ 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™ server to All.

Set the logging level of Avaya Breeze™ 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™ 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™ servers out of debug mode.

Launch Avaya Breeze™ Terminal

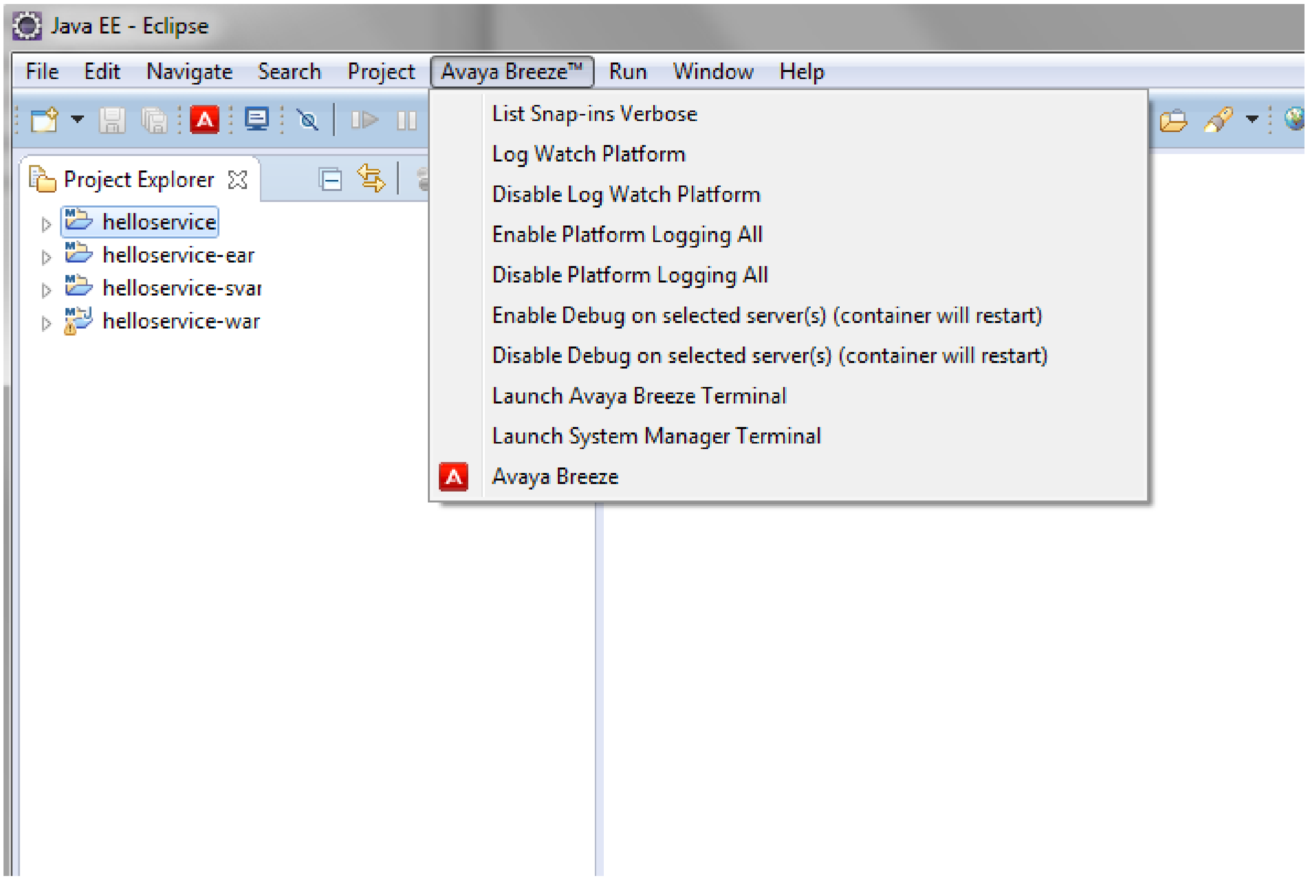
Launch the Avaya Breeze™ terminal by using the configured user name and password.

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™.



List Snap-ins Verbose	This action will list the services deployed on Avaya Breeze™ server.
Log Watch Platform	The Avaya Breeze™ 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™ server to ALL (i.e., enable fine, finer and finest logging levels). Default logging level for Avaya Breeze™ 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™ 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™ servers out of debug mode.
Launch Avaya Breeze™ Terminal	Launch the Avaya Breeze™ 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.

Converting an older project to Avaya Breeze™ 3.1

Before you begin

- 3.0 SDK is successfully installed.
- JDK 1.7 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-3.1.1</artifactId>
               <version>3.1.1.1.311103</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 1.6 dependency with 1.7 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-2.0</artifactId>
        <version>2.0.2.1.202105</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-3.1.1</
artifactId>
             <version>>3.1.1.1.311103</version>
```

   b. Replace the highlighted part:

```
<dependency>
            <groupId>com.avaya.zephyr.zephyrCommon.xsds</groupId>
            <artifactId>archive-xsd</artifactId>
            <version>2.0.2.1.202105</version>
            <exclusions>
            <exclusion>
            <artifactId>smc-api</artifactId>
            <groupId>com.avaya.zephyr.services.smc</groupId>
            </exclusion>
            </exclusions>
```

```
    </dependency>

with
<version>>3.1.1.1.311103</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>
```

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. Then, 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 (i.e., it is a called party service). Information on configuring services to be invoked on incoming or outgoing calls will be 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 done similarly, 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, that is, 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™ API Call drop() method is invoked before a call has been answered, Avaya Breeze™ generates and sends a block message to the caller's endpoint. In general, endpoints can handle Avaya Breeze™'s block message in different ways. Some endpoints might play a tone such as reorder, while other endpoints might provide silence 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 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 http://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 *handle* is a name, telephone number, or E.164 number, and the *domain* is either a host or 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
        }
}
```

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);
...
```

Outgoing calls

The Avaya Breeze™ 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™ 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.

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 generally 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 or not. 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, an Avaya Aura® Media Server is required in the system. The AAMS 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 whole 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").

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.

Snap-in start/stop from Avaya Breeze™ Element Manager

A snap-in can be programmed to allow the system administrator to start and stop it for a specified cluster from the Avaya Breeze™ Element Manager. The default behavior is for a snap-in to not have this start/stop capability. The system administrator can use the **Start** and **Stop** buttons on the Avaya Breeze™ Service Management page to start or stop snap-ins that are programmed with this feature. For additional information, see *Administering Avaya Breeze™*.

Snap-ins that are defined as stoppable in `properties.xml` have the start/stop feature.

Example

```
<attribute name="SnapInStoppableName">
  <displayName>SnapInStoppableName</displayName>
  <helpInfo>The existence will enable Snap-In Stop and Start capabilites. This cannot be
changed.</help>
  <global>false</global>
  <validation name="booleanType">
    <type>STRING</type>
    <pattern>(?i)true|false</pattern>
  </validation>
  <admin_visible>false</admin_visible>
  <admin_changeable>false</admin_changeable>
  <factory>
    <value>True</value>
    <user_changeable>false</user_changeable>
  </factory>
</attribute>
```

Inserting and removing Avaya Aura® Media Server

Avaya Breeze™ has the ability to 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 the 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 needn't be concerned about removing the media server after the media operation completes. After several seconds go by without any active media operations, the media server will automatically be removed. The list of media operations that will cause the media server to be inserted include: play, record, send or collect digits, speech search, and Voice XML dialog.

A mechanism exists whereby the default media server inclusion policy of "AS_NEEDED" can be overridden to instead be set to "INCLUDED". With this setting, the media server will immediately be added to the call and will not be removed until the setting is changed to “AS_NEEDED”. There are 2 cases where a snap-in will want to use the “INCLUDED” media server inclusion policy:

1. The process of inserting / removing the media server causes a momentary disruption in the audio path. In some cases, a snap-in may find it preferable to leave the 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 the 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™ 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, the media server will be inserted when the next media operation is invoked.

Remove media server MediaServerInclusion.AS_NEEDED:

This default policy causes Avaya Breeze™ to remove the media server from the stream when no media operations are active. If the previous policy was INCLUDED, the 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 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™ currently supports two modes of service invocation: calling party services and called party 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.

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 (t`estService-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>
```

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 leves 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 nex 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>
        <type>String</type>
        </validation>
        <admin_visible>true</admin_visible>
        <factory>
            <value>Hello from Breeze</value>
        </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: Either String or encryptedString
- Admin visible: Determines whether the attribute appears in the System Manager administration UI.
- Factory: this section describes the default values for the attribute
  - Value: the default value for the attribute (i.e., the value used if a Service profile does not define its own value for the attribute)

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 Configiration 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 don't 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>
```

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 adminstrator can select a Snap-in URL from the Service URL column on the Cluster Admininstration 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 v“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]Admini 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’re 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™ 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.

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

```
@TheServiceLifeCycle
public class MyServiceLifeCycle implements ServiceLifeCycle
{

    @Override
     public void init()
    {
         // 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™ 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 10 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 20 megabytes, the line would look as follows: `<log_space>20MB<log_space>`. The log for your service will be located in `/var/log/Avaya/services/ServiceName/`.

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. The following example of an alarm definition is 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>
        <severitybbmajor<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");
        }
```

Avaya Breeze™ 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 (e.g., 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.

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.

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);
```

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:

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.

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
```

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

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.

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://Breeze-Cluster-IP/services/<snap-in name>/index.jsp` and the request gets to Avaya Breeze™, 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™ 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.

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 of log file. After log size reaches the maximum value, 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 on 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.

.<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.

Chapter 3: Avaya Breeze™ connectors

Avaya Breeze™ connectors

Avaya Breeze™ 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.
- Clickatell SMS Connector to send SMS messages.

Scopia connector

Introduction

The Scopia Connector enables programmatic access from Avaya Breeze™ 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 Scopia® Management application through the Scopia XML-based API. Please note that the XML-based Scopia® 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 things in Scopia® Management. You optionally can also configure Scopia® Management to allow the use of HTTPS. A summary of how to configure Scopia® 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 on page 70.

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 Scopia® Management. This account is a standard user account from the point of view of Scopia® Management, but it will be the account used to authenticate API connections from Avaya Breeze™. 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 Scopia® 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 on page 70.

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.
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™.
Password for API Access to Conference Service	This is the password previously configured in Scopia® Management.
Service Prefix for Scheduling Conferences	This is the prefix assigned to the Meeting Type that was previously configured in Scopia® 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 Scopia® Management application. You can use either HTTP or HTTPS. If you use HTTPS, you'll need to be sure that Scopia® Management has been configured to allow TLS connections. Also, depending on the configuration of Scopia® 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 Scopia® 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 Scopia® 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™ 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™ host and the Scopia® 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™ host has a time later than the time on the Scopia® Management host, the start time of the conference will be delayed.
- If the Avaya Breeze™ host has a time earlier than the time on the Scopia® Management host, the conference may fail to be scheduled, depending on the version of Scopia® Management that is being run. Some versions of Scopia® 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 Scopia® Management.

Email connector

Introduction

The Email SMTP Connector is a convenience service provided by Avaya Breeze™ 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™ API suite.

As its name suggests, the Email SMTP Connector communicates only via SMTP (the Simple Mail Transfer Protocol), so this connector can be used only to send email. The current versions of Email API and Email SMTP Connector support only single-part plain text emails (i.e., no HTML body, no multi-part, no attachments).

Overview

A “connector” is a Avaya Breeze™ service that provides connectivity to some application that is external to Avaya Breeze™. 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™ 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 on page 73.

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.
Use SMTPS to Host #1	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.
Use SMTPS to Host #2	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

Table continues…

Field	Description
	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.
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™ 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.

Clickatell SMS connector

Introduction

The Clickatell SMS Connector is a convenience service provided by Avaya Breeze™ that gives a service writer an easy way to send an SMS to 1 or more recipients. As its name suggests, the Clickatell SMS Connector communicates with Clickatell (see https://www.clickatell.com/) to send SMS messages, so you must have an account with Clickatell to use the Clickatell SMS Connector.

Overview

A “connector” is a Avaya Breeze™ service that provides connectivity to some application that is external to Avaya Breeze™. The Clickatell SMS Connector communicates with Clickatell over HTTP/S, using the HTTP/S API described on Clickatell’s web site.

A service can send a request to the Clickatell SMS Connector using the Avaya Breeze™ SMS API. The response to that request indicates the status of the request as reported by Clickatell, obtained as part of the exchange between the Clickatell SMS Connector and Clickatell. This status can indicate, for example, that an SMS was delivered to a recipient or a network, an SMS delivery is pending (e.g., queued for delivery), or some error.

Description

To use the Clickatell SMS Connector, you will need to create an account with Clickatell for an HTTP API connection. You should consider your requirements for sending SMS when choosing the type of account to create. You should also be aware of regional regulations or conventions for SMS. You will need this information from the account:

- User name
- Password
- Api-id (in the Clickatell portal, look under Manage my Products; note that this is *not* the client ID)

You will also need to define at least 1 Sender ID (e.g., sender's phone number).

This information must be provided to the Clickatell SMS Connector, either through the SMS API or by configuration.

Configuring the Clickatell SMS connector

About this task

Configure attributes for the Clickatell SMS 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 **ClickatellSmsConnector** from the **Service** pulldown
4. Fill out the Attributes Configuration page for the Clickatell SMS Connector according to the Clickatell SMS connector field descriptions on page 78

Clickatell SMS connector field descriptions

Field	Description
SMS Provider Service Base URI	This is the base URI used to form the HTTP/S requests to Clickatell. You probably will not need to modify this field, unless you want to use HTTP instead of HTTPS, or if you want to point to some kind of emulator for testing.

Table continues…

Field	Description
Default User Name for the SMS Account	This is the user name for the Clickatell account you created. If your service provides a user name in the SMS API (via SmsRequest.setUserName), that user name will override this field (meaning that this field can be left blank).
Default Password for the SMS Account	This is the password for the Clickatell account you created. If your service provides a password in the SMS API (via SmsRequest.setPassword), that password will override this field (meaning that this field can be left blank).
Default api_id	This is the api_id from a connection in your Clickatell account. If your service provides an api_id in the SMS API (via SmsRequest.setAccountId), that api_id will override this field (meaning that this field can be left blank).
Test mode enabled?	When you are first testing your service that uses the Clickatell SMS 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 Clickatell SMS 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 Clickatell SMS connector from your service

Use the SMS API (which is a part of the larger Avaya Breeze™ API suite) to request that an SMS be sent to 1 or more recipients (up to a maximum of 100 recipients in one request). The SMS API is a general API designed to operate with potentially many flavors of SMS connector, so some of the method names in the API will not appear to directly correlate with a specific SMS connector. For example, the general accountId in the API (accessed with the setAccountId and getAccountId methods) is used to convey the api_id specific to the Clickatell SMS connector. The API consists of:

- A request object (SmsRequest) that you obtain from a factory and populate with the needed information (e.g., list of recipients, text message). The “send” method on the request sends the request to the SMS 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. Note that the message can include Unicode characters (this is supported natively for the java String object).
- A listener object that you implement (to the SmsListener interface) if you are interested in viewing the response from the SMS Connector.
- A response object (SmsResponse) that holds the response from the SMS 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 a list of recipients (the “d” parameter), a text message (the “m” parameter), an api_id (the “a” parameter), the sender’s phone number (the “s” parameter), user name (the “u” parameter), and password (the “p” parameter).

```
    protected void sendRequest(final HttpServletRequest request, final
HttpServletResponse response)
            throws IOException
    {
        final String destination = request.getParameter("d");
        String message = request.getParameter("m");
        final String accountId = request.getParameter("a");
        int status = HttpServletResponse.SC_INTERNAL_SERVER_ERROR;
        final String[] destinations = destination.split(",");
        final List <String> recipients = new ArrayList <String> ();
        for (String d : destinations)
        {
            recipients.add(d);
        }

        final SmsRequest smsreq = SmsFactory.createSmsRequest(recipients, message);
        smsreq.setAccountId(accountId);
        final String sender = request.getParameter("s");
        if (sender != null)
        {
            smsreq.setSender(sender);
        }
        final String username = request.getParameter("u");
        if (username != null)
        {
            smsreq.setUserName(username);
        }
        final String pwd = request.getParameter("p");
        if (pwd != null)
        {
            smsreq.setPassword(pwd);
        }

        String msg = null;
        try
        {
            SmsListener listener = new SmsTesterListener();
            smsreq.setListener(listener);
            smsreq.send();
            status = HttpServletResponse.SC_OK;
        }
        catch (final Exception e)
        {
            msg = "caught exception trying to send request to connector: " + e.toString();
        }

        if (msg != null)
        {
            final PrintWriter w = response.getWriter();
            response.setContentType("text/plain");
            w.println(msg);
        }
        response.setStatus(status);
    }
```

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

- Get an SmsRequest using the factory.`final SmsRequest smsreq = SmsFactory.createSmsRequest(recipients, message);`

- Create a listener and setting that listener in the SmsRequest.`SmsListener listener = new SmsTesterListener(id, recipients, message);` , `smsreq.setListener(listener);`
- Send the request to the connector. `smsreq.send();`

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

```
public final class SmsTesterListener implements SmsListener
{
    private Logger logger = Logger.getLogger(SmsTesterListener.class);
    private final long startTime;

    public SmsTesterListener()
    {
        this.startTime = System.currentTimeMillis();
    }

    @Override
    public void responseReceived(final SmsResponse response)
    {
        final long elapsed = System.currentTimeMillis() - startTime;
        final StringBuilder sb = new StringBuilder();
        sb.append("SmsTesterListener responseReceived: ")
                .append("status = ").append(response.getStatus())
                .append(", detail = ").append(response.getDetail())
                .append(", delivered =
").append(Joiner.on(",").join(response.getDeliveredRecipients()))
                .append(", pending =
").append(Joiner.on(",").join(response.getPendingRecipients()))
                .append(", failed =
").append(Joiner.on(",").join(response.getFailedRecipients()));
        logger.info(sb.toString());
        try
        {
            final Duration duration = DatatypeFactory.newInstance().newDuration(elapsed);
            logger.info("SmsTesterListener: duration = " + duration.toString());
        }
        catch (final Exception e)
        {
            // can't log duration
        }
    }
}
```

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 “delivered” list holds those recipients for which Clickatell reports a delivered status. Note that “delivered” might not mean that the SMS was delivered to the device – it could mean the SMS was delivered to a network gateway, for example. The “pending” list holds those recipients for which Clickatell reports a “queued” status. The “failed” recipients list holds those recipients for which Clickatell reports a failure to deliver.

The response shows the status obtained from Clickatell shortly after the request to send an SMS was sent to Clickatell. To find the status at some later point in time, you will need to log in to the Clickatell portal and run a report.

Note that some time is needed to retrieve the status for each recipient. If a request is sent to the SMS connector with a large number of recipients, the callback to the listener could be a few minutes after sending the request.

Chapter 4: Callable services

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 800 number.

Callable service snap-in configuration

In a call intercept scenario, Avaya Breeze™ is provisioned as a "sequenced application" that is then 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™ is then able to invoke the appropriate snap-in(s) based on the provisioned service profile for the user.

In a callable service scenario, Avaya Breeze™ is not provisioned as a sequenced application. Instead, a routing policy is configured to route calls with a particular pattern to Avaya Breeze™. Session Manager then 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. The configuration on Avaya Breeze™ for a callable snap-in is done in exactly the same way as a called party call intercept snap-in, except that there can be only one snap-in in the service profile.

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

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™ 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™ 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.

Chapter 5: 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 in 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);
        }
    }
}
```

Scaling

Avaya Breeze™ is designed to run in a clustered configuration. There may be up to 5 Avaya Breeze™ instances running the same set of services. So a service should not rely on resources on a specific Avaya Breeze™ instance such as configuration files or databases. Instead, these resources should be placed on a remote system that all of the Avaya Breeze™ instances have access to.

Additional resources

For further information about the Avaya Breeze™ APIs, check the Avaya Breeze™ 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™ FAQ and Troubleshooting for Snap-in Developers* and the Avaya Breeze™ forum on Avaya DevConnect.