Progress | Artix

WSDLGen Guide

Version 5.6, December 2011

Third Party Acknowledgments:

Progress Artix ESB for C++ v5.6 incorporates Xalan v2.3.1technologies from the Apache Software Foundation (http://www.apache.org). Such Apache technologies are subject to the following terms and conditions: The Apache Software License, Version 1.1. Copyright (C) 1999-2002 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Ant", "Xerces," "Xalan," "Log 4J," and "Apache Software Foundation" must not be used to: endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org/. Xalan was originally based on software copyright (c) 1999, Lotus Development Corporation., http://www.lotus.com. Xerces was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com.

Progress Artix ESB for C++ v5.6 incorporates Xerces C++ v2.4 technology from the Apache Software Foundation (http://www.apache.org). Such Apache technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 - Copyright (c) 1999-2001 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 incorporates Apache Xerces v2.5.0 technology from the Apache Software Foundation ((http://www.apache.org). Such Apache technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 - Copyright (c) 1999-2002 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress Artix ESB for C++ v5.6 incorporates Xerces C++ v1.7 technology from the Apache Software Foundation (http://www.apache.org). Such Apache technology is subject to the following terms and conditions: The Apache Software License, Version 1.1. - Copyright (c) 1999-2004 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xalan" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARIS-

ING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, Lotus Development Corporation., http://www.lotus.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress Artix ESB for C++ v5.6 incorporates Apache Velocity v1.3 technology from the Apache Software Foundation (http://www.apache.org). Such Apache technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 - Copyright (c) 2000-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgement: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

4. The names "The Jakarta Project", "Velocity", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", "Velocity" nor may "Apache" appear in their names without prior written permission of the Apache Group.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 incorporates Log4J v1.2.6 technology from the Apache Software Foundation (http://www.apache.org). Such Apache technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 - Copyright (C) 1999 The Apache Software Foundation. All rights reserved. Redistribution and use in

source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "log4j" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLU DING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

(a) Progress Artix ESB for C++ v5.6 incorporates JDOM Beta 9 technology from JDOM. Such technology is subject to the following terms and conditions: Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <request_AT_jdom_DOT_org>. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <request_AT_jdom_DOT_org>. In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This

product includes software developed by the JDOM Project (http://www.jdom.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos. THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the JDOM Project and was originally created by Jason Hunter <jhunter_AT_jdom_DOT_org> and Brett McLaughlin <brett_AT_jdom_DOT_org>. For more information on the JDOM Project, please see <http://www.jdom.org/>

Progress Artix ESB for C++ v5.6 incorporates IBM-ICU v2.6 and IBM-ICU v2.6.1 technologies from IBM. Such technologies are subject to the following terms and conditions: Copyright (c) 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder. All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Progress Artix ESB for C++ v5.6 incorporates John Wilson MinML v1.7 technology from John Wilson. Such technology is subject to the following terms and conditions: Copyright (c) 1999, John Wilson (tug@wilson.co.uk). All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by John Wilson. The name of John Wilson may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY JOHN WILSON ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JOHN WILSON BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 incorporates SourceForge - NET-SNMP v5.0.7 technology from SourceForge and Networks Associates Technology, Inc. Such technology is subject to the following terms and conditions: Various copyrights apply to this package, listed in various separate parts below. Please make sure that you read all the parts. Up until 2001, the project was based at UC Davis, and the first part covers all code written during this time. From 2001 onwards, the project has been based at SourceForge, and Networks Associates Technology, Inc hold the copyright on behalf of the wider Net-SNMP community, covering all derivative work done since then. An additional copyright section has been added as Part 3 below also under a BSD license for the work contributed by Cambridge Broadband Ltd. to the project since 2001. An additional copyright section has been added as Part 4 below also under a BSD license for the work contributed by Sun Microsystems, Inc. to the project since 2003. Code has been contributed to this project by many people over the years it has been in development, and a full list of contributors can be found in the README file under the THANKS section. ---- Part 1: CMU/UCD copyright notice: (BSD like) ----- Copyright 1989, 1991, 1992 by Carnegie Mellon University. Derivative Work - 1996, 1998-2000. Copyright 1996, 1998-2000 The Regents of the University of California. All Rights Reserved. Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU and The Regents of the University of California not be used in advertising or publicity pertaining to distribution of the software without specific written permission. CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CMU OR THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR

IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. ---- Part 2: Networks Associates Technology, Inc copyright notice (BSD) ----- Copyright (c) 2001-2003, Networks Associates Technology, Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: *Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.* Neither the name of the Networks Associates Technology, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ---- Part 3: Cambridge Broadband Ltd. copyright notice (BSD) ----- Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:*Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.* The name of Cambridge Broadband Ltd. may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE

OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ---- Part 4: Sun Microsystems, Inc. copyright notice (BSD) -----Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved. Use is subject to license terms below. This distribution may include materials developed by third parties. Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the fol-

lowing conditions are met:* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.* Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ---- Part 5: Sparta, Inc copyright notice (BSD) -----Copyright (c) 2003-2005, Sparta, Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.* Neither the name of Sparta, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ---- Part 6: Cisco/BUPTNIC copyright notice (BSD) ----- Copyright (c) 2004, Cisco, Inc and Information Network Center of Beijing University of Posts and Telecommunications. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Cisco, Inc, Beijing University of Posts and Telecommunications, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS

PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. ---- Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD) ----- Copyright (c) Fabasoft R&D Software GmbH & Co KG, 2003 oss@fabasoft.com Author: Bernhard Penz. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries, brand or product names may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 incorporates OpenSSL/SSLeay v0.9.8i technology from OpenSSL.org. Such Technology is subject to the following terms and conditions: LICENSE ISSUES ==========

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License ----------

/*

==========

====

given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Progress Artix ESB for C++ v5.6 incorporates Bouncycastle v1.3.3 cryptographic technology from the Legion Of The Bouncy Castle (http://www.bouncycastle.org). Such Bouncycastle 1.3.3 cryptographic technology is subject to the following terms and conditions: Copyright (c) 2000 - 2006 The Legion Of The Bouncy Castle (http://www.bouncycastle.org). Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Progress Artix ESB for C++ v5.6 incorporates PCRE 7.8 from PCRE for the purpose of providing a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. Such technology is subject to the following terms and conditions: PCRE LICENCE. PCRE is a library of functions to support regular expressions whose syntax and semantics are as close as possible to those of the Perl 5 language. Release 7 of PCRE is distributed under the terms of the "BSD" licence, as specified below. The documentation for PCRE, supplied in the "doc" directory, is distributed under the same terms as the software itself. The basic library functions are written in C and are freestanding. Also included in the distribution is a set of C++ wrapper functions. THE BASIC LIBRARY FUNCTIONS. Written by: Philip Hazel. Email local part: ph10. Email domain: cam.ac.uk. University of Cambridge Computing Service, Cambridge, England. Copyright (c) 1997-2008 University of Cambridge All rights reserved. THE C++ WRAPPER FUNCTIONS. Contributed by: Google Inc. Copyright (c) 2007-2008, Google Inc. All rights reserved. THE "BSD" LICENCE. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of the University of Cambridge nor the name of Google Inc. nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 incorporates mcpp v2.6.4 from Kiyoshi Matsui. Such technology is subject to the following terms and conditions: Copyright (c) 1998, 2002-2007 Kiyoshi Matsui kmatsui@t3.rim.or.jp All rights reserved. This software including the files in this directory is provided under the following license. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress Artix ESB for C++ v5.6 contains IBM Licensed Materials Copyright IBM Corporation 2010 (IBM 32-bit Runtime Environment for AIX, Java Technology Edition v 1.6.0 SR9 FP2).

Updated: December 5, 2011

Contents

List of Figures

Preface

What is Covered in This Book

This book describes how to use the WSDLGen command-line utility to generate code from a WSDL contract. As well as describing the standard WSDLGen code generating templates, the book explains how to develop custom templates, which you can then use to generate Artix applications implemented in C++.

Who Should Read This Book

This book is aimed primarily at C++ developers who are interested in using code generation to accelerate the process of implementing Web service applications.

This book might also be of some interest to build engineers who need to generate Makefiles based on the content of WSDL contracts.

The Artix Documentation Library

For information on the organization of the Artix library, the document conventions used, and where to find additional resources, see Using the Artix Library

CHAPTER 1

Using WSDLGen

This chapter explains how to use the standard templates provided with WSDLGen to generate sample applications in C++.

In this chapter

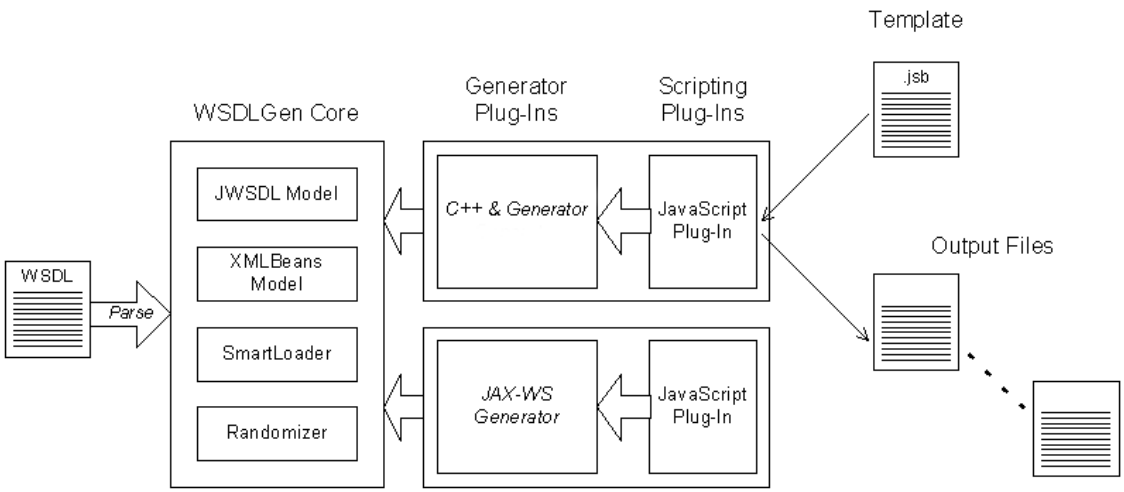
This chapter discusses the following topics:

WSDLGen Architecture

Overview

Figure 1 provides an overview of the WSDLGen code generator architecture.

Figure 1: *WSDLGen Code Generator Architecture*



WSDLGen core

The WSDLGen core consists of a pluggable framework—for loading generator and scripting plug-ins—as well as providing a core set of libraries, which are made available to the loaded plug-ins.

The core set of libraries includes the following object models, which can represent the parsed contents of the WSDL contract, as follows:

- *JWSDL model*—a model that recognizes the standard elements of a WSDL contract, identifying each type of WSDL element with a Java class.
- *XMLBeans model*—a model that recognizes the elements of an XML schema definition. This model is used to represent the `types` section of a WSDL contract (where the parameter data types are defined).

For more details about the core parsers, see "Parser Overview" on page 60.

The WSDLGen core also includes additional utilities, as follows:

- *SmartLoad utility*—provides the capability to load template files from a well-known location (a search path for SmartLoad can be specified in the WSDLGen configuration file).
- *Randomizer utility*—can be used to generate random parameter data. This is useful for generating sample application code.

Generator plug-ins

Because the WSDLGen core provides only the abstract framework for code generation, it is always necessary to specify a particular generator plug-in when you invoke WSDLGen. Currently, the following generators are provided:

- C++ generator plug-in
- JAX-WS generator plug-in.

C++ generator plug-in

You must specify the C++ generator in order to generate code for an Artix C++ application.

JAX-WS generator plug-in

The JAX-WS generator, used to generate code for an Artix JAX-WS Java application, is not available in the Artix ESB for C++ product. The JAX-WS generator is available in the Artix ESB for Java product.

Scripting plug-ins

The WSDLGen architecture has been designed so that it is possible to support additional template languages by adding a plug-in to the core. Currently, WSDLGen supports only the JavaScript language.

JavaScript plug-in

JavaScript (also known as ECMAScript) is an object-based scripting language that has a syntax similar to C or Java. Unlike object-oriented languages, however, JavaScript is not a strongly-typed language.

The JavaScript plug-in enables you to write code-generating templates in the JavaScript language. The choice of JavaScript as the template language has no impact on the choice of generated language: you can use JavaScript templates to generate code in C++, Java or any other language.

Standard templates

WSDLGen provides a standard suite of templates that take a WSDL contract and generate a sample *Artix application* in C++ based on the interfaces defined in the contract.

Custom templates

It is also possible for you to develop your own custom templates. An easy way to get started with developing custom templates is to take one of the standard WSDLGen templates and modify it for your own requirements—see "Developing Basic Templates" on page 37 for details.

Generating Code with the wsdlgen Utility

Syntax of wsdlgen

The `wsdlgen` command-line utility has the following syntax:

```
wsdlgen [-G ApplicationType] [-T TemplateID]*
    [-C ConfigFile] [-D Name=Value]* WSDLFile
```

Where a pair of square brackets, `[]`, denotes an optional part of the syntax and the asterix character, `*`, implies that the preceding option can be repeated 0 or more times.

You must specify the location of a valid WSDL contract file, *`WSDLFile`*. You can also supply the following options:

`-G` *`ApplicationType`*	Specifies the type of application to generate. The following application types are defined by default: • `cxx`—for generating C++ code.
`-T` *`TemplateID`*	Each application type defines a set of template IDs, which can be used as shortcuts to invoke particular template scripts. For details, see "Generating C++ code" on page 29.
`-C` *`ConfigFile`*	Specifies the location of the WSDLGen configuration file, *`ConfigFile`*. If this option is not set, `wsdlgen` reads the default configuration file (located in `%IT_WSDLGEN_CFG_FILE%` on Windows and `$IT_WSDLGEN_CFG_FILE` on UNIX).
`-D` *`Name=Value`*	Specifies the value, *`Value`*, of a JavaScript property, *`Name`*. See also "Variables defined at the command line" on page 28.

Generating code from a specific template (or templates)

You can specify explicitly which templates to run, by invoking the `wsdlgen` utility with the `-T` option. For example, suppose you have a WSDL contract file, `hello_world.wsdl`, and you wish to generate a sample implementation of the `Greeter` port type. You could invoke the `artix wsdlgen` utility as follows:

```
wsdlgen
    -G
    com.iona.artix.tools.wsdlgen.classicgenerator.CPPAndJaxRPCGen
    erator -D portType=Greeter
    -T templates\cxx\ArtixCxxImplH.jsb
    -T templates\cxx\ArtixCxxImplCxx.jsb
    hello_world.wsdl
```

Variables defined at the command line

The following JavaScript variables can be set at the command line, using the `-D` option of the `artix wsdlgen` command:

- `portType`—local name of the port type for which code is generated.
- `bindingName`—local name of the binding for which code is generated.
- `serviceName`—local name of the service for which code is generated.
- `portName`—name of the port for which code is generated.

In particular, you can set the following combinations of these variables at the command line in order to select a particular service and port:

- `serviceName` and `portName`—generate code for the specified service and port.
- `serviceName`—generate code for the specified service and the first port of that service.
- `portType`—generate code for the first service, port, and binding associated with the specified port type.
- `bindingName`—generate code for the first service and port associated with the specified binding.
- *None specified*—generate code for the first service and port in the WSDL contract.

Generating C++ code

When generating C++ code from the standard templates, it is usually simpler to use the `-G cxx -T TemplateID` syntax. For example, to generate a sample implementation of the `Greeter` port type from the `hello_world.wsdl` file, you could invoke the `wsdlgen` utility as follows:

```
wsdlgen
    -G cxx -T impl -D portType=Greeter hello_world.wsdl
```

When called with `-G cxx`, the `-T TemplateID` switch supports the following template IDs:

`impl`	For the given *`PortType`* port type (specified by the `portType` property), generate the files *`PortType`*`Impl.h` and *`PortType`*`Impl.cxx` that implement *`PortType`*. Also, generate stub code and type files for the port type.
`server`	For the given *`PortType`*, generate a file, *`PortType`*`ServerSample.cxx`, that implements the `main()` function for a standalone server. Also, generate stub code and type files for the port type.
`client`	For the given *`PortType`*, generate a file, *`PortType`*`ClientSample.cxx`, that invokes all of the operations in the *`PortType`* port type. Also, generate stub code and type files for the port type.
`plugin`	For the given *`PortType`*, generate all of the files required for a plug-in implementation of the server. The resulting plug-in can then be deployed into an Artix container (see *Developing Artix Applications in C++* for more details).
`all`	Specifying `-G cxx -T all` is equivalent to specifying `-G cxx -T impl -T plugin -T client`.
`make`	Generate a Makefile for the C++ application. This option must be used in combination with one or more of the following options `-T plugin`, `-T server`, `-T client`, or `-T all`.

C++ Templates

Overview

WSDLGen provides a variety of standard templates that you can use to generate sample application code directly from a WSDL contract. These templates are located in the *ArtixInstallDir*/tools/templates directory.

Invoking the C++ templates

To invoke a C++ template directly, use the -G option to load the CPPAndJaxRPCGenerator generator class. For example, to generate code from a single C++ template, *TemplateFile*, enter a command like the following:

```
artix wsdlgen
    -G
    com.iona.artix.tools.wsdlgen.classicgenerator.CPPAndJaxRPCGen
    erator
    -T TemplateFile
    WSDLFile
```

C++ templates

Table 1 lists the WSDLGen templates that can be used to generate C++ examples.

Table 1: *WSDLGen Templates for Generating C++ Code*

C++ Template File	Description
ArtixCxxActivatorCxx.jsb	Generate the implementation of a *service activator* class (to use in conjunction with a container plug-in). When a service is deployed in an Artix container, the service activator makes it possible to start and stop the service at runtime using the it_container_admin utility.
ArtixCxxActivatorH.jsb	Generate the header file for the service activator class.
ArtixCxxClientMain.jsb	Generate a sample C++ client.
ArtixCxxDeployDescr.jsb	Generate an XML deployment descriptor for deploying a plug-in into the Artix container.
ArtixCxxImplCxx.jsb	Generate an outline servant implementation for the port type specified by the portType property.
ArtixCxxImplH.jsb	Generate the header file for the servant implementation.

Table 1: *WSDLGen Templates for Generating C++ Code*

C++ Template File	Description
`ArtixCxxMakefile.jsb`	Generate a sample Makefile.
`ArtixCxxPlugin.jsb`	Generate a sample plug-in implementation (for deploying into an Artix container).
`ArtixCxxPluginScript.jsb`	Generate a script that starts an Artix container process and deploys the plug-in into the container.
`ArtixCxxServerMain.jsb`	Generate a sample server `main()` function (for a standalone application).
`ArtixCxxStubTypes.jsb`	Generate stub code for specified port type specifed by the `portType` property.

WSDLGen Configuration File

Overview

The `wsdlgen` utility has its own configuration file, which is defined in XML format. This configuration file enables you to customize WSDLGen by:

- Setting JavaScript variables.
- Setting SmartLoader paths.
- Defining generator profiles.

Default location

The WSDLGen configuration is stored at the following default location:

```
ArtixInstallDir/tools/etc/wsdlgen.cfg
```

Setting JavaScript variables

You can initialize JavaScript variables from the WSDLGen configuration file, as shown in Example 1.

Example 1: *Setting JavaScript Variables in the Configuration File*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdlgen>
    <defines>
        <foo>fooValue</foo>
        <!-- ... -->
    </defines>
    ...
</wsdlgen>
```

Where the `defines` element can contain any number of entries of the form `<VariableName>Value</VariableName>`. Each configuration entry of this form is equivalent to including the following JavaScript code at the top of your template:

```
var VariableName = "Value";
```

Setting SmartLoader paths

You can define a search path for the smart loader utility in the WSDLGen configuration file by adding a sequence of `path` elements inside an enclosing `paths` element, as shown in Example 2.

Example 2: *Setting SmartLoader Paths in the Configuration File*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdlgen>
    <paths>
        <path>/home/fflintstone/.wsdlgen</path>
        <path>/usr/local/templates/wsdlgen</path>
        <!-- ... -->
    </paths>
    ...
</wsdlgen>
```

When searching for scripts included through the smart loader mechanism, WSDLGen searches the directories listed in the `paths` element. For more details about the smart loader utility, see "smartLoader utility" on page 47.

Defining generator profiles

You can define your own *generator profiles* in the WSDLGen configuration file. A generator profile enables you to customize the combination of templates that are invoked when you enter a WSDLGen command of the form `wsdlgen -G` *`ApplicationType`* `-T` *`TemplateID`*. This is typically useful, if you are developing your own WSDLGen templates.

Example 3 shows the general outline of a generator profile in the WSDLGen configuration file.

Example 3: *Defining a Generator Profile in the Configuration File*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdlgen>
    ...
1   <profiles>
2       <ApplicationType>
3           <generator>GeneratorClass</generator>
4           <TemplateID>
5               <template>TemplatePath</template>
                ...
            </TemplateID>
            ...
        </ApplicationType>
```

Example 3: *Defining a Generator Profile in the Configuration File*

```
    </profiles>
    ...
</wsdlgen>
```

The preceding profile configuration can be explained as follows:

1. The `profiles` element contains one or more arbitrarily-named profile elements, *`ApplicationType`*.
2. An *`ApplicationType`* element represents a single generator profile. You can call this element anything you like: the `wsdlgen` utility automatically searches for the *`ApplicationType`* element when you specify it using the `-G` *`ApplicationType`* option.
3. The `generator` element specifies the name of the generator plug-in class to use for this profile.
4. A *`TemplateID`* identifies a combination of templates that can be called in a single batch. This element can have an arbitrary name: the `wsdlgen` utility automatically searches for the *`TemplateID`* element when you specify it using the `-T` *`TemplateID`* option.

   You can define multiple *`TemplateID`* elements within each profile.
5. Within each template combination, use the `template` element to specify the location of a single template. Typically, you would specify the absolute pathname of the template. You can also substitute environment variables from the operating system, using the syntax, `$`*`VARIABLE_NAME`*`$`.

Examples of generator profiles

For some examples of generator profiles, see the profiles for `cxx` in the default configuration file, *`ArtixInstallDir`*`/tools/etc/wsdlgen.cfg`.

Unsupported XML Schema Types

Overview

Currently, not all XML schema types are supported by the WSDLGen code generator. The type support depends on which kind of code you are generating, as follows:

- Unsupported types in C++ mappings.

Unsupported types in C++ mappings

The following XML schema types are currently not supported by the C++ code generator:

- Primitive types: `xs:nonPositiveInteger`, `xs:duration`, `xs:NMTokens`, `xs:IDREF`, `xs:IDREFS`, `xs:ENTITY`, `xs:ENTITIES`, `xs:NOTATION`.
- `xs:list`
- `xs:union`
- `xs:group`
- Types derived by extension or restriction.
- Occurrence constraints on `xs:sequence` and `xs:choice`.
- Anonymous types.
- `xs:choice`
- Nillable types.

CHAPTER 2

Developing Basic Templates

This chapter provides an introduction to the subject of writing your own templates for generating code in C++.

In this chapter

This chapter discusses the following topics:

Writing Custom Templates

Overview

The simplest approach to take when writing a custom template is to take one of the WSDLGen samples and modify it to your own requirements. This chapter aims to provide you with enough information to understand the sample templates and to use the WSDLGen programming interfaces effectively.

Running a custom template

To generate code using a custom template, specify the template file to the `wsdlgen` utility using the `-T` command-line option and specify the relevant generator class using the `-G` command-line option.

Bilingual files

WSDLGen templates are written in a special file format known as a *bilingual file* and identified by the `.jsb` file suffix. The bilingual file format enables you to freely mix the JavaScript language and the target language together in the one file. For details, see "Bilingual Files" on page 39.

Predefined objects

To provide you with convenient access to data and objects derived from the WSDL contract, WSDLGen creates predefined objects in JavaScript. For example, the `wsdlModel` object provides access to a complete parse tree of the WSDL contract (using the JWSDL API).

For details, see "Predefined Objects" on page 44.

Built-in APIs

A few different APIs are provided for writing templates, as follows:

- *WSDLGen API for Artix C++*—utility functions for generating Artix C++ code from WSDL.
- *WSDLGen randomizer*—a random data generator, used internally by WSDLGen to generate random parameter values.
- *JWSDL API*—a WSDL parser based on the JWSDL standard. See "The JWSDL Parser" on page 69 for details.
- *XMLBeans API*—an XML schema parser. See "The XMLBeans Parser" on page 77 for details.

Bilingual Files

Overview

The basic purpose of a JavaScript template in WSDLGen is to generate code in a *target language* (such as C++). Consequently, if a code generating template was written in pure JavaScript, it would contain a large number of print directives to produce the required target code. In practice, this style of coding quickly leads to templates that are virtually illegible (you might be familiar with this sort of problem in the context of HTML-generating servlet code).

To solve this difficulty, WSDLGen introduces the concept of a *bilingual file* for developing code-generating templates. The basic idea of the bilingual file is that a set of escape sequences enable you to switch back and forth between the generating language and the target language. Example 4 shows a sample outline of such a bilingual file, with one section of the file (enclosed between `[***` and `***]`) expressed in the target language.

Example 4: *Sample Outline of a Bilingual File.*

```
// JavaScript Bilingual File
openOutputFile(PathName)

// Put JavaScript code here...
...

[***
    // Put TargetLanguage code here...
    ...
***]

closeOutputFile()
```

Opening and closing the output file

A bilingual file must be associated with an output destination. You can specify an output file for the generated code by calling the following function in your script (typically, at the start of the template):

```
openOutputFile(PathName)
```

Where *PathName* specifies the path to the generated output file. On UNIX platforms, an alternative form of the `openOutputFile()` function is available, which lets you set file permissions on the output file:

```
openOutputFile(PathName, Permissions)
```

Where *Permissions* is a string value formatted in the same way as a standard `chmod` permission string. For example, the string, `u=rwx,g=rx,o=x`, would give full permissions to the owner, read and execute permissions to the group, and execute permission to all others. For full details of the permission string syntax, enter `man chmod` at the command line.

You can close the output file by calling the following function (typically, at the end of the template):

```
closeOutputFile()
```

The call to `openOutputFile()` establishes an association between the destination file, *PathName*, and the blocks of generated code written in the target language. All of the generated code is sent to the file, *PathName*, specified by the `openOutputFile()` function.

Note: If `openOutputFile()` is not called, the output is directed to standard out by default.

Output text delimiters

Blocks of generated code are delimited by the output text delimiters shown in Table 2.

Table 2: *Character Sequences for Delimiting Output Text*

Character Sequence	Description
`[***`	Beginning of a code block written in the target language.

Table 2: *Character Sequences for Delimiting Output Text*

Character Sequence	Description
`***]`	End of the code block written in the target language.

Escaping within output text

Within the scope of the output text delimiters, you can escape back to JavaScript using the escape characters shown in Table 3.

Table 3: *Escape Characters Used in Output Text*

Escape Sequence	Description
`$`*`VarName`*`$`	Substitute a JavaScript variable, *`VarName`*, embedding it in a line of output text—see "Variable escape" on page 41.
`@`*`JavaScript`*	Escape to a line of JavaScript—see "Line escape" on page 42.

Variable escape

Within the scope of the output text delimiters, you can substitute the value of a JavaScript variable using the dollar sign, `$`, as an escape character. To make the substitution, enclose the JavaScript variable name between two dollar signs, `$`*`VarName`*`$`.

For example, if `intfName` is a JavaScript variable that holds a WSDL port type name, you could declare a Java class to implement this port type using the following fragment of bilingual file.

```
// JavaScript Bilingual File
openOutputFile(PathName)

[***
public class $intfName$Impl implements java.rmi.Remote {
***]

// More script (not shown)...
...
closeOutputFile()
```

The implementation class name is derived by adding the `Impl` suffix to the porty type name. For example, if generating code for the `Greeter` port type, `$intfName$Impl` would expand to `GreeterImpl`.

Line escape

Within the scope of the output text delimiters, you can escape to a line of JavaScript code by putting the *at* symbol, `@`, at the start of a line (as the first non-whitespace character).

For example, the following bilingual file generates a Java function, `ListInterfaceOps()`, that lists all of the operations in the current WSDL interface.

```
// JavaScript Bilingual File
...
openOutputFile(PathName)

[***
    ...
    public static void ListInterfaceOps() {
        System.out.println("Operation is one of: ");
        @for (var i = 0; i < numOps; i++) {
        System.out.println("  $operations[i].getName()$");
        @}
    }
}
***]

closeOutputFile()
```

Unlike the variable escape mechanism, `$VarName$`, the line escape does *not* produce any output text as a side effect of its execution. While the line enclosing a variable escape sequence, `$VarName$`, is implicitly enclosed in a print statement, the line escaped by the *at* symbol, `@`, is not printed.

Escaping the escape characters

Occasionally, you might need to output the dollar, `$`, and at sign, `@`, character literals inside the scope of an output text block. For this purpose, WSDLGen defines the `$dollar$` and `$at$` variables, which resolve to literal dollar, `$`, and literal *at*, `@`, inside an output text block.

For example, you could insert the `$` and `@` character literals into your output code, as shown in the following example:

```
// JavaScript Bilingual File
```

```
...
[***
    cout << "Here is a contrived example," << endl;
    cout << "that shows how to get the $dollar$ and $at$"
         << " literals into your output." << endl;
***]
```

Predefined Objects

Overview

The programming interface provided by WSDLGen includes a number of predefined JavaScript objects. Some of these predefined objects are simple variables (for example, `intfName`, containing the name of the current port type), whilst others provide access to particular APIs (for example, `wsdlModel`, which provides access to the JWSDL parser API).

List of predefined objects

Table 4 shows the list of JavaScript objects predefined by WSDLGen.

Table 4: *Predefined JavaScript Objects*

JavaScript Object	Description
`bindingName`	Local part of the binding name for which code is generated. You can set this variable when you invoke the command (see "Variables defined at the command line" on page 28).
`cxxIntfName`	A name derived from `intfName` by replacing any dot characters, `.`, with underscores, `_`. For example, `simple.simpleIntf` would become `simple_simpleIntf`.
`cxxNamespace`	The C++ namespace in which to define the generated implementation classes. Its value is derived from the WSDL target namespace.
`cxxServiceName`	A name derived from `serviceName` by replacing any dot characters, `.`, with underscores, `_`. For example, `simple.simpleService` would become `simple_simpleService`.
`intfName`	A name derived from the port type name, `portType`, by dropping the `PortType` suffix (if any).
`jsModel`	A wrapper for the `wsdlModel` object.
`operations[]`	An array of operation objects, of `javax.wsdl.Operation` type. See "JWSDL Parser Classes" on page 72 for details.

Table 4: *Predefined JavaScript Objects*

JavaScript Object	Description
`parametersList`	An instance of the utility class, `com.iona.wsdlgen.common.ParametersList`. This object enables you to obtain a list of parts and faults for every WSDL operation.
`portName`	Port name for which code is generated. You can set this variable when you invoke the `artix wsdlgen` command (see "Variables defined at the command line" on page 28).
`portType`	Local part of the port type name for which code is generated. You can set this variable when you invoke the `artix wsdlgen` command (see "Variables defined at the command line" on page 28).
`randomizer`	An instance of a WSDLGen utility that generates random numbers. The WSDLGen templates use this object to generate random parameters.
`schemaModel`	An instance of the `org.apache.xmlbeans.SchemaTypeLoader` class, which provides access to an XML schema parser. See "The XMLBeans Parser" on page 77 for details.
`serviceName`	Local part of the service name for which code is generated. You can set this variable when you invoke the `artix wsdlgen` command (see "Variables defined at the command line" on page 28).
`smartLoader`	An instance of a WSDLGen utility that imports JavaScript or bilingual files from a well-known location. The search path for the smart loader can be specified in the WSDLGen configuration file.
`tns`	The namespace of the port type, binding, and service elements. Specifically, this variable contains the value of the `targetNamespace` attribute from the `definitions` element in the WSDL contract.
`wsdlFile`	The location of the WSDL contract file.

Table 4: *Predefined JavaScript Objects*

JavaScript Object	Description
`wsdlModel`	An instance of the `javax.wsdl.Definition` class, which provides access to a JWSDL parser. See "Parsing WSDL and XML" on page 59 for details.

WSDL and schema models

The following objects represent the roots of the WSDL model and the XML schema model respectively:

- `wsdlModel`
- `schemaModel`

These parser objects provide a complete model of the WSDL elements and XML schema types defined in the WSDL contract.

Typically, it is not necessary to use these APIs in a basic template. For more advanced applications, however, see "Parsing WSDL and XML" on page 59 for details about the parser APIs.

operations[] array

An array of operation objects representing all of the operations in the `portType` port type. The operation objects are instances of `javax.wsdl.Operation`, which is part of the JWSDL API.

For example, you can print out the names of all the operations in the `portType` port type as follows:

```
// JavaScript Bilingual File
...
for (var i=0; i < operations.length; i++) {
[***
    System.out.println("Operation["+i+"] name = "
        + $operations[i].getName()$
    );
***]
}
```

For more details about the `javax.wsdl.Operation` class, see "JWSDL Parser Classes" on page 72.

parametersList object

The `parametersList` object provides a method, `getPartsAndFaults()`, that provides access to all of the message parts and faults associated with a particular WSDL operation.

For example, to obtain the parts and faults associated with the `i`th operation of the current WSDL interface, make the following JavaScript call:

```
var partsAndFaults = parametersList.getPartsAndFaults(
       portType + operations[i].getName()
   )
```

Where the argument to `getPartsAndFaults()` is a key, consisting of a port type name concatenated with an operation name.

By calling `partsAndFaults.parts()[k]`—where `k` lies in the range `0` to `partsAndFaults.parts().length`—you can obtain a `PartHolder` object, which holds the following items:

- `partsAndFaults.parts()[k].getPart()`—returns the `javax.wsdl.Part` object that represents the current part.
- `partsAndFaults.parts()[k].getDirection()`—returns one of the following direction flag values: `DIRECTION_IN`, or `DIRECTION_OUT`.

By calling `partsAndFaults.faults()[k]`—where `k` lies in the range `0` to `partsAndFaults.faults().length`—you can obtain a `FaultHolder` object, which holds the following items:

- `partsAndFaults.faults()[k].getName()`—returns the fault name.
- `partsAndFaults.faults()[k].getParts()`—returns the array of `javax.wsdl.Part` objects contained in the fault.

smartLoader utility

The *smart loader utility* provides a way of including files located relative to a well-known directory (or directories). For example, if you are implementing a custom template, you could include the contents of the file, `CustomUtils/MyUtilities.js`, at the start of your template by calling `smartLoad()` as follows:

```
# JavaScript Bilingual File
smartLoad("CustomUtils/MyUtilities.js");
...
```

Where the included file, `CustomUtils/MyUtilities.js`, is located under one of the directories listed in the `paths` element in the WSDLGen configuration file. Example 5 shows an example of a configuration file that specifies two path directories, with each directory enclosed in a `path` element. The directories are searched in the order in which they appear in the configuration file.

Example 5: *Smart Loader Path in the WSDLGen Configuration File*

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdlgen>
    <paths>
        <path>/home/fflintstone/.wsdlgen</path>
        <path>/usr/local/templates/wsdlgen</path>
        <!-- ... -->
    </paths>
    ...
</wsdlgen>
```

Generating C++ Code

Overview

This section provides a brief overview of the most important WSDLGen functions for generating C++ code. The following topics are described:

- Mangling identifiers.
- Generating print calls.
- Generating operation calls in a C++ consumer.
- Functions for generating C++ implementations.
- Generating an implementation header.
- Generating a C++ implementation class.

Mangling identifiers

Table 5 summarizes the functions that you can use to mangle identifiers according to the Artix C++ mangling rules.

Table 5: *Functions for Mangling C++ Identifiers*

Function	Description
`cxxMangleMethodName ( opName )`	Return a mangled method name, according to the Artix C++ name mangling rules.
`cxxMangleTypeName( typeName )`	Return a mangled type name, according to the Artix C++ name mangling rules.
`cxxMangleVarName( varName )`	Return a mangled object name or class name, according to the Artix C++ name mangling rules.

Generating print calls

Table 6 summarizes the WSDL functions that you use to generate C++ functions that print operation parameter values.

Table 6: *Functions for Generating C++ Print Calls*

Function	Description

`cxxPrintMethodSig(`
`  portType,`
`  opName,`
`  ignoreDirection`
`)`
	Generate the signature of a C++ function that prints out the parameters of the operation, *opName*, in the interface, *portType*. The *ignoreDirection* parameter specifies which kind of parameter *not* to print. The *ignoreDirection* parameter can have one of the following values: `DIRECTION_IN`, `DIRECTION_OUT`, `DIRECTION_INOUT`.

`cxxPrintParts(`
`  portType,`
`  opName,`
`  lvl,`
`  ignoreDirection`
`)`
	Generate the body of a C++ function that prints out the parameters of the operation, *opName*, in the interface, *portType*.

`cxxPrintCall(`
`  portType,`
`  opName,`
`  ignoreDirection`
`)`
	Generate a C++ function call that calls the print function generated by the preceding utility functions.

Generating operation calls in a C++ consumer

Table 7 summarizes the WSDLGen functions that you use to generate a WSDL operation call in the C++ language.

Table 7: *Functions for Generating an Operation Call in C++*

Function	Description

`cxxPopulateParts(`
`  portType,`
`  opName,`
`  lvl,`
`  isClient`
`)`
	When the *isClient* parameter is equal to `true`, populate each of the request parameters (*in* and *inout* parameters) with random data.

Table 7: *Functions for Generating an Operation Call in C++*

Function	Description

`cxxMethodCall(`
`  portType,`
`  opName`
`)`
	Call the operation, *opName*.

The functions in Table 7 take the following arguments:

- *portType* is the local name of the port type on which the operation is defined;
- *opName* is a `javax.wsdl.Operation` instance that represents the operation in the WSDL model;
- *lvl* specifies how many levels of indentation are applied to the generated code.
- *isClient* is a boolean flag that indicates whether the function is being called to generate client-side code (`true`) or server-side code (`false`).

Example 6 shows how to use the preceding functions to generate the operation calls in a Web service client. The code iterates over every operation in the current port type, generating code to declare and initialize the parameters and then call the operation.

Example 6: *Generating Operation Calls in C++*

```
// JavaScript Bilingual File
...
for (var i = 0; i < operations.length; i++) {
[***
{
    $cxxPopulateParts(portType, operations[i].getName(), 1,
   true)$

    $cxxMethodCall(portType, operations[i].getName())$

    $cxxPrintCall(portType, operations[i].getName(),
   DIRECTION_IN)$
}

***]
}
```

For example, if the preceding script is run against the `hello_world.wsdl` file, it generates the C++ code shown in Example 7.

Example 7: *Generated Operation Calls in C++*

```
// C++

{
    IT_Bus::String theResponse;

    client->sayHi(theResponse);
}

{
    IT_Bus::String me;
    IT_Bus::String theResponse;

    me = "Curry";

    client->greetMe(me, theResponse);
}
```

Functions for generating C++ implementations

Table 8 summarizes the WSDLGen functions that you use to generate an implementation class in the C++ language.

Table 8: *Functions for Generating a C++ Implementation*

Function	Description
`artixCxxOperSig( Prefix, PortTypeName, Op, IndentLevel, Trailing )`	Return the signature of the operation, *`Op`*, in the port type, *`PortTypeName`*. This function can be used in various contexts; that is, either in the header file or the C++ implementation file. The Prefix string—which should be in the format *`CxxNamespace`*`::`*`ClassName`*`::`—allows you to prefix the function signature with the name of the implementation class. The trailing string, *`Trailing`*, is appended to the end of the generated signature.

Table 8: *Functions for Generating a C++ Implementation*

Function	Description
`cxxOperImpl( portType, opName, echoParams, genFlag )`	Generates code to populate the *out* parameters of the operation, *opName*, in the interface, *portType*. Normally, the parameters are populated with random values. However, if you specify the *echoParams* flag to be `true`, any parameters declared both *in* and *out* will echo the incoming value back to the caller. The `genFlag` flag must be a string with the value `cxx`.
`cxxPopulateParts( portType, opName, lvl, isClient )`	When the *isClient* parameter is equal to `false`, populate each of the reply parameters (*inout* and *out* parameters) with random data.

Generating an implementation header

Example 8 shows a script that uses the preceding functions to generate an implementation header file. The script iterates over all of the operations in the current port type, generating a function declaration for each one.

Example 8: *Generating a C++ Implementation Header*

```
// JavaScript Bilingual File
...
var UpperInfName = cxxIntfName.toUpperCase()
[***
#ifndef IT_Bus_$cxxNamespace$_$UpperInfName$IMPL_INCLUDED_
#define IT_Bus_$cxxNamespace$_$UpperInfName$IMPL_INCLUDED_

#include "$cxxIntfName$Server.h"

namespace $cxxNamespace$
{
    class $cxxIntfName$Impl : public $cxxIntfName$Server
    {
      public:

        $cxxIntfName$Impl(IT_Bus::Bus_ptr bus);
```

Example 8: *Generating a C++ Implementation Header*

```
        virtual ~$cxxIntfName$Impl();

        virtual IT_Bus::Servant*
        clone() const;


@var numOps = operations.length
@for (var i = 0; i < numOps; i++) {

    $artixCxxOperSig("", portType, operations[i], 2, ";\n")$
@}

    };
}

#endif
***]
```

For example, if the preceding script is run against the `hello_world.wsdl` file, it generates the C++ header file shown in Example 9.

Example 9: *Generated C++ Implementation Header*

```
#ifndef
   IT_Bus_COM_IONA_HELLO_WORLD_SOAP_HTTP_GREETERIMPL_INCLUDED_
#define
   IT_Bus_COM_IONA_HELLO_WORLD_SOAP_HTTP_GREETERIMPL_INCLUDED_

#include "GreeterServer.h"

using namespace COM_IONA_HELLO_WORLD_SOAP_HTTP;

namespace COM_IONA_HELLO_WORLD_SOAP_HTTP
{
    class GreeterImpl : public GreeterServer
    {
      public:

        GreeterImpl(
            IT_Bus::Bus_ptr bus
        );

        virtual ~GreeterImpl();
```

Example 9: *Generated C++ Implementation Header*

```
        virtual IT_Bus::Servant*
        clone() const;

        void
        sayHi(
            IT_Bus::String& theResponse
        ) IT_THROW_DECL((IT_Bus::Exception));

        void
        greetMe(
            const IT_Bus::String& me,
            IT_Bus::String& theResponse
        ) IT_THROW_DECL((IT_Bus::Exception));

    };
}

#endif //IT_SIMPLE_SERVICE_IMPL_INCLUDED_
```

Generating a C++ implementation class

Example 10 shows a script that uses the functions from Table 8 on page 52 to generate a C++ implementation class. The script iterates over all of the operations in the current port type, generating a member function for each one.

Example 10: *Generating a C++ Implementation Class*

```
// JavaScript Bilingual File
...
[***
#include "$cxxIntfName$Impl.h"
#include <it_cal/cal.h>
#include <it_cal/iostream.h>
#include <it_bus/to_string.h>

IT_USING_NAMESPACE_STD
using namespace $cxxNamespace$;
using namespace IT_Bus;

$cxxIntfName$Impl::$cxxIntfName$Impl(IT_Bus::Bus_ptr bus) :
   $cxxIntfName$Server(bus)
{

}
```

Example 10: *Generating a C++ Implementation Class*

```
$cxxIntfName$Impl::~$cxxIntfName$Impl()
{

}

IT_Bus::Servant*
$cxxIntfName$Impl::clone() const
{
    return new $cxxIntfName$Impl(get_bus());
}

***]

var numOps = operations.length
for (var i = 0; i < numOps; i++) {
[***
void
$cxxPrintMethodSig(portType, operations[i].getName(),
   DIRECTION_OUT)$
{
   cout << "\n\nOperation $operations[i].getName()$ received:\n"
   << endl;
    $cxxPrintParts(portType, operations[i].getName(), 1,
   DIRECTION_OUT)$
}

$artixCxxOperSig(cxxIntfName + "Impl::", portType,
   operations[i], 0, "")$
{
$cxxPrintCall(portType, operations[i].getName(), DIRECTION_OUT)$
$cxxOperImpl(portType, operations[i], true, "cxx")$
}

***]

}
```

For example, if the preceding script is run against the `hello_world.wsdl` file, it generates the C++ implementation class shown in Example 11.

Example 11: *Generated C++ Implementation Class*

```
#include "GreeterImpl.h"
#include <it_cal/cal.h>
#include <it_cal/iostream.h>
#include <it_bus/to_string.h>

IT_USING_NAMESPACE_STD
using namespace $cxxNamespace$;
using namespace IT_Bus;

GreeterImpl::GreeterImpl(
    IT_Bus::Bus_ptr bus
) : GreeterServer(bus)
{
    // complete
}

GreeterImpl::~GreeterImpl()
{
   // complete
}

IT_Bus::Servant*
$intfName$Impl::clone() const
{
    return new $intfName$Impl(get_bus());
}

void
GreeterImpl::sayHi(
    IT_Bus::String& theResponse
) IT_THROW_DECL((IT_Bus::Exception))
{
    theResponse = IT_Bus::String("Curry");
}

void
GreeterImpl::greetMe(
    const IT_Bus::String& me,
    IT_Bus::String& theResponse
) IT_THROW_DECL((IT_Bus::Exception))
{
    theResponse = me;
}
```

CHAPTER 3

Parsing WSDL and XML

This chapter introduces you to the subject of parsing WSDL using the low-level APIs, JWSDL and Apache XMLBeans. The higher-level WSDLGen API is built on top of these basic parsing APIs.

In this chapter

This chapter discusses the following topics:

Parser Overview

Overview

The parsing APIs that underly WSDLGen are taken from the following open source products:

- WSDL4J (reference implementation of the JWSDL standard),
- Apache XMLBeans.

These two parsers provide alternative views of the WSDL contract. The JWSDL model is useful for parsing WSDL artifacts, such as port types, bindings, and services. The XMLBeans model, on the other hand, is an XML schema parser, which is more useful for parsing the XML schema types defined in the WSDL contract.

JWSDL

JWSDL is a Java API for parsing WSDL contracts. This API is being developed under the Java Community Process, JSR 110. A copy of the JWSDL specification and complete Javadoc for the JWSDL API can be downloaded from the following location:

http://jcp.org/en/jsr/detail?id=110

Apache XMLBeans

Apache XMLBeans is an open source API for parsing XML schemas. It is useful for parsing the contents of the `schema` elements in a WSDL contract. The home page for the XMLBeans project is:

http://xmlbeans.apache.org/

The complete Javadoc for XMLBeans v2.2.0 is available at the following location:

http://xmlbeans.apache.org/docs/2.2.0/reference/index.html

Rhino

Rhino is a Java implementation of JavaScript that includes the capability to map Java APIs into JavaScript (the *scripting Java* feature). In the context of WSDLGen, this capability of Rhino is exploited to make both the JWSDL API and the XMLBeans API available in JavaScript (these APIs are originally specified in Java only).

Due to the strong similarity between Java syntax and JavaScript syntax, the mapped APIs are remarkably intuitive to use from within JavaScript. For details about how this mapping works, see:

http://www.mozilla.org/rhino/ScriptingJava.html

Basic Parsing

Overview

This section discusses some basic topics in parsing WSDL contracts. In particular, you need to be aware of how the contract style (document/literal wrapped or RPC/literal) affects how you parse a WSDL port type.

In this section

This section contains the following subsections:

The WSDL and XML Schema Models

Overview

WSDLGen enables JavaScript programs to access the JWSDL API and the XMLBeans API from by defining the following JavaScript objects:

- `wsdlModel`—the root of the JWSDL parser model.
- `schemaModel`—the root of the XMLBeans parser model.

These two objects are pushed into JavaScript using the Rhino Java-to-JavaScript mapping feature.

wsdlModel instance

To access the JWSDL API from within JavaScript, use the `wsdlModel` object, which is an instance of the `javax.wsdl.Definition` class mapped to JavaScript.

The JWSDL `Definition` class represents the top level element of the WSDL contract (see "JWSDL Parser Classes" on page 72). For example, you can use the `wsdlModel` object to obtain a list of all the port types in the contract as follows:

```
// JavaScript

var portTypeMap = wsdlModel.getPortTypes()
var portTypeArr = portTypeMap.values().toArray()

// Iterate over the list of port types
for each (pt in portTypeArr) {
    ... // Do something with the port type, pt.
}
```

schemaModel instance

To access the XMLBeans API from within JavaScript, use the `schemaModel` object, which is an instance of the `org.apache.xmlbeans.SchemaTypeLoader` class mapped to JavaScript.

The XMLBeans `SchemaTypeLoader` class enables you to find the XML schema types and elements defined within the `wsdl:types` element in the WSDL contract (see "XMLBeans Parser Classes" on page 79). For example, you can use the `schemaModel` object to obtain an element named `{http://xml.iona.com/wsdlgen/demo}testParams`, as follows:

```
// JavaScript

var TARG_NAMESPACE = "http://xml.iona.com/wsdlgen/demo"
var elQName = new javax.xml.namespace.QName(TARG_NAMESPACE,
   "testParams")

var el = schemaModel.findElement(elQName)
```

Parsing Document/Literal Wrapped Style

Overview

This subsection describes how to parse a WSDL contract that is written in *document/literal wrapped* style. The document/literal wrapped style is distinguished by the fact that it uses single part messages. Each part is defined to be a sequence type, whose constitutent elements represent operation parameters.

Characteristics of the document/literal wrapped style

A given operation, *`OperationName`*, must be defined as follows, in order to conform to the document/literal wrapped style of interface:

- *Input message*—the `message` element that represents the operation's input message must obey the following conditions:
  - → The message contains just *a single part*.
  - → The part references an element (not a type) and the element must be named, *`OperationName`*.
- *Input element*—the *`OperationName`* element must be defined as a `sequence` complex type, where each element in the sequence represents a distinct input parameter.
- *Output message*—the `message` element that represents the operation's output message must obey the following conditions:
  - → The message contains just *a single part*.
  - → The part references an element (not a type) and the element must be named, *`OperationName`*`Response`.
- *Output element*—the *`OperationName`*`Response` element must be defined as a `sequence` complex type, where each element in the sequence represents a distinct output parameter.

Sample WSDL contract

Example 12 shows an example of a WSDL contract defining an operation, `testParams`, that conforms to document/literal wrapped style.

Example 12: *Operation Defined in Document/Literal Style*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ... >
    <wsdl:types>
        <schema targetNamespace="..."
```

Example 12: *Operation Defined in Document/Literal Style*

```
                xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="testParams">
                <complexType>
                    <sequence>
                      <element name="inInt" type="xsd:int"/>
                     <element name="inoutInt" type="xsd:int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="testParamsResponse">
                <complexType>
                    <sequence>
                       <element name="inoutInt" type="xsd:int"/>
                      <element name="outFloat" type="xsd:float"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </wsdl:types>
    <message name="testParams">
        <part name="parameters" element="tns:testParams"/>
    </message>
    <message name="testParamsResponse">
        <part name="parameters"
             element="tns:testParamsResponse"/>
    </message>
    <wsdl:portType name="BasePortType">
        <wsdl:operation name="testParams">
            <wsdl:input  message="tns:testParams"
                         name="testParams"/>
            <wsdl:output message="tns:testParamsResponse"
                         name="testParamsResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    ...
</definitions>
```

Parsing RPC/Literal Style

Overview

This subsection describes how to parse a WSDL contract that is written in *RPC/literal* style. The RPC/literal style is distinguished by the fact that it uses multi-part messages (one part for each parameter).

Characteristics of the RPC/literal style

A given operation, `OperationName`, must be defined as follows, in order to conform to the RPC/literal style of interface:

- *Input message*—the `message` element that represents the operation's input message must obey the following conditions:
  - → The message can contain multiple parts, where each part represents a distinct input parameter.
  - → Each part references a type (not an element).
- *Output message*—the `message` element that represents the operation's output message must obey the following conditions:
  - → The message can contain multiple parts, where each part represents a distinct output parameter.
  - → Each part references a type (not an element).

Sample WSDL contract

Example 13 shows an example of a WSDL contract defining an operation, `testParams`, that conforms to RPC/literal style.

Example 13: *Operation Defined in RPC/Literal Style*

```
<definitions ...>
    ...
    <message name="testParams">
        <part name="inInt" type="xsd:int"/>
        <part name="inoutInt" type="xsd:int"/>
    </message>
    <message name="testParamsResponse">
        <part name="inoutInt" type="xsd:int"/>
        <part name="outFloat" type="xsd:float"/>
    </message>
    ...
    <portType name="BasePortType">
        <operation name="testParams">
            <input message="tns:testParams" name="testParams"/>
            <output message="tns:testParamsResponse"
                    name="testParamsResponse"/>
        </operation>
    ...
</definitions>
```

The JWSDL Parser

Overview

This section contains a partial summary of the JWSDL parser API. Only the parts of the API that you would need for generating application code are described here. For a complete description of the API, see JSR 110.

In this section

This section contains the following subsections:

Overview of the WSDL Model

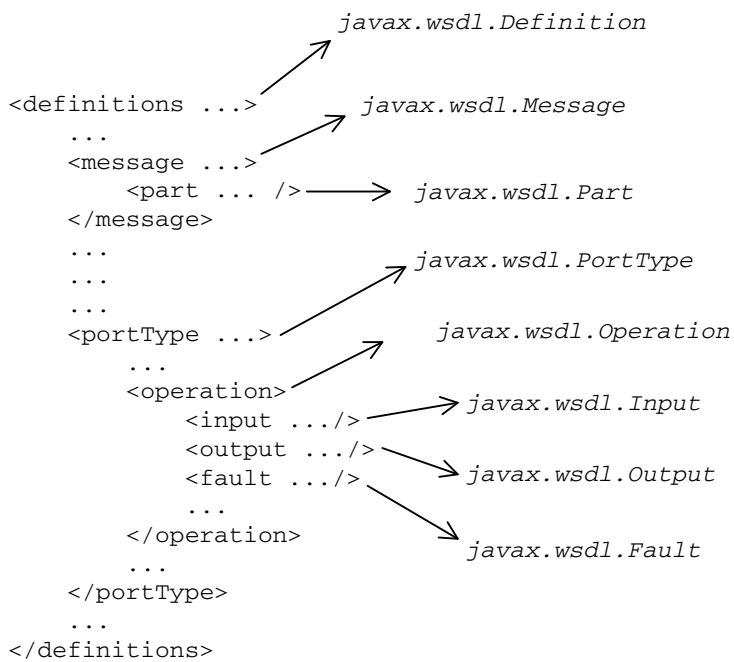
Overview

This section provides a partial overview of the WSDL model supported by the JWSDL parser. We focus here on the subset of the JWSDL API that is useful for generating application code from a WSDL contract. Hence, the discussion omits the API for parsing `wsdl:binding` and `wsdl:service` elements. The API for parsing the `wsdl:portType` element, which is essential for generating application code, is described here.

JWSDL classes required for parsing a port type

Figure 2 provides an overview of the JWSDL classes required for parsing a WSDL port type, showing how each JWSDL class corresponds to an element of the original WSDL contract.

Figure 2: *JWSDL Classes for Parsing a Port Type*

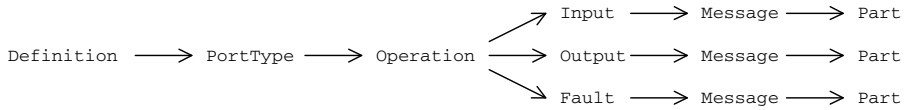


Generally, each JWSDL class is named after the element it represents. Note, however, that the class representing the `definitions` element is called `Definition`, *not* `Definitions`.

Node hierarchy

Each JWSDL class in the nodal hierarchy provides methods to access the WSDL elements it contains or, in some cases, references. Figure 3 shows the most convenient paths you can take to navigate down the node hierarchy when parsing a WSDL port type.

Figure 3: *Navigating the JWSDL Node Hierarchy*



Once you get down as far as a `javax.wsdl.Part` node, you can retrieve the QName of the element (or type) that represents a particular operation argument. To progress further with the parsing, you need to switch to the XMLBeans API, which enables you to parse the XML schema encoding the argument data (see "The XMLBeans Parser" on page 77).

JWSDL Parser Classes

Overview

This subsection summarizes the JWSDL parser classes that are likely to prove most useful when attempting to parse a port type in the context of generating code.

The following JWSDL classes are summarized here:

- javax.wsdl.Definition
- javax.wsdl.PortType
- javax.wsdl.Operation
- javax.wsdl.Input
- javax.wsdl.Output
- javax.wsdl.Fault
- javax.wsdl.Message
- javax.wsdl.Part

Useful Java utility classes

A number of Java utility classes are used with the JWSDL parser API (for example, aggregate types such as `java.util.List`). For your convenience, a brief overview of these utility classes is provided in Appendix A on page 85.

For the complete Javadoc API, consult the following Javadoc reference:

http://java.sun.com/j2se/1.5.0/docs/api

javax.wsdl.Definition

The `javax.wsdl.Definition` class represents a `wsdl:definition` element (top level of a WSDL contract). The most useful methods from the `javax.wsdl.Definition` class are shown in Table 9.

Table 9: *Methods from the javax.wsdl.Definition Class*

Method Signatures	Description
`java.util.Map getPortTypes()`	Get the `portType` elements defined in this `definition` element.

`javax.wsdl.PortType getPortType(`
`  javax.xml.namespace.QName name`
`)`
	Get the `portType` element with the specified name.

Table 9: *Methods from the javax.wsdl.Definition Class*

Method Signatures	Description
`java.util.Map getAllPortTypes()`	Get the `portType` elements defined in this `definition` element and those in any imported `definition` elements down the WSDL tree.
`java.util.Map getImports()`	Get a map of lists containing all the imports defined here.
`java.util.Map getImports(String namespaceURI)`	Get the list of imports for the specified `namespaceURI`.
`java.util.Map getNamespaces()`	Get all namespace associations in this definition.
`String getNamespace(String prefix)`	Get the namespace URI associated with this prefix.
`String getPrefix(String namespaceURI)`	Get a prefix associated with this namespace URI.
`String getTargetNamespace()`	Get the target namespace in which the WSDL elements are defined.

javax.wsdl.PortType

The `javax.wsdl.PortType` class represents a `wsdl:portType` element. The most useful methods from the `javax.wsdl.PortType` class are shown in Table 10.

Table 10: *Methods from the javax.wsdl.PortType Class*

Method Signatures	Description
`java.util.List getOperations()`	Get the operations defined in this port type.

`javax.wsdl.Operation getOperation(`
`  String name,`
`  String inputName,`
`  String outputName`
`)`
	Get the operation with the specified name, `name`. If the operation name is overloaded, you can optionally use the `inputName` (the name of the operation's `input` element) and/or the `outputName` (the name of the operation's `output` element) to disambiguate. Otherwise, set `inputName` and `outputName` to `null`.

`javax.xml.namespace.QName getQName()`	Returns the name of the port type.
`boolean isUndefined()`	True if this port type is not defined.

javax.wsdl.Operation

The `javax.wsdl.Operation` class represents a `wsdl:operation` element. The most useful methods from the `javax.wsdl.Operation` class are shown in Table 11.

Table 11: *Methods from the javax.wsdl.Operation Class*

Method Signatures	Description
`javax.wsdl.Input getInput()`	Get this operation's `input` element.
`javax.wsdl.Output getOutput()`	Get this operation's `output` element.
`java.util.Map getFaults()`	Get this operation's `fault` elements.
`javax.wsdl.Fault getFault(String name)`	Get the `fault` with the specified name.
`String getName()`	Returns the name of the operation.
`boolean isUndefined()`	True if the operation is undefined.

javax.wsdl.Input

The `javax.wsdl.Input` class represents a `wsdl:input` element. The most useful methods from the `javax.wsdl.Input` class are shown in Table 12.

Table 12: *Methods from the javax.wsdl.Input Class*

Method Signatures	Description
`javax.wsdl.Message getMessage()`	Get the input message element.
`String getName()`	Return the name of the `input` element (if any).

javax.wsdl.Output

The `javax.wsdl.Output` class represents a `wsdl:output` element. The most useful methods from the `javax.wsdl.Output` class are shown in Table 13.

Table 13: *Methods from the javax.wsdl.Output Class*

Method Signatures	Description
`javax.wsdl.getMessage()`	Get the output message element.
`String getName()`	Return the name of the `output` element (if any).

javax.wsdl.Fault

The `javax.wsdl.Fault` class represents a `wsdl:fault` element. The most useful methods from the `javax.wsdl.Fault` class are shown in Table 14.

Table 14: *Methods from the javax.wsdl.Fault Class*

Method Signatures	Description
`javax.wsdl.Message getMessage()`	Get the fault message element.
`String getName()`	Return the name of the `fault` element (if any).

javax.wsdl.Message

The `javax.wsdl.Message` class represents a `wsdl:message` element. The most useful methods from the `javax.wsdl.Message` class are shown in Table 15.

Table 15: *Methods from the javax.wsdl.Message Class*

Method Signatures	Description
`java.util.Map getParts()`	Get a map of the message's parts, where the key is a part name and the value is a `javax.wsdl.Part` object.
`javax.wsdl.Part getPart(String name)`	Get the part specified by `name`.
`javax.xml.namespaceQName getQName()`	Get the qualified name of this `message` element.
`boolean isUndefined()`	True if this message element is undefined.

javax.wsdl.Part

The `javax.wsdl.Part` class represents a `wsdl:part` element. The most useful methods from the `javax.wsdl.Part` class are shown in Table 16.

Table 16: *Methods from the javax.wsdl.Part Class*

Method Signatures	Description
`javax.xml.namespace.QName getElementName()`	Get the element node referred to by the part's `element` attribute (if any).
`javax.xml.namespace.QName getTypeName()`	Get the type node referred to by the part's type attribute (if any).
`String getName()`	Get the name of the part.

The XMLBeans Parser

Overview

This section contains a partial summary of the XMLBeans parser API, which can be used to parse the parameter data from WSDL operations at runtime. For a complete description of the API, see the XMLBeans 2.2.0 Javadoc.

In this section

This section contains the following subsections:

Overview of the XMLBeans Parser

Overview

This section provides a partial overview of the classes in the XMLBeans parser. The XMLBeans parser actually supports two different kinds of schema model: a static model and a dynamic (runtime) model. The static model is created by generating a set of Java classes that represent the elements of an XML schema. The dynamic model, on the other hand, does not require any Java classes to be generated and can parse any XML schema at runtime.

The section focusses on describing the dynamic (runtime) model.

XMLBeans classes needed to parse XML schema

The following XMLBeans classes are essential for the runtime parsing of XML data:

- `org.apache.xmlbeans.SchemaTypeLoader`—a class that enables you to look up schema types and schema global elements by name.
- `org.apache.xmlbeans.SchemaGlobalElement`—a class that represents elements defined *directly* inside the `xsd:schema` element (in contrast to elements defined at a nested level in the schema, which are known as *local elements*).

  Note: The main difference between a global element and a local element is that a global element can be defined to be a member of a substitution group, whereas a local element cannot. In addition, the elements referenced within a `wsdl:part` element would normally be global elements.

- `org.apache.xmlbeans.SchemaType`—the class that represents a schema type.
- `org.apache.xmlbeans.SchemaProperty`—a class that represents a summary of the elements that *share the same name* within a complex type definition.

  Note: XML schema allows you to define an element with the same name *more than once* inside a complex type declaration.

XMLBeans Parser Classes

Overview

This subsection summarizes the most important XMLBeans parser classes, which you are likely to use while parsing an XML schema type in WSDLGen.

The following XMLBeans classes are summarized here:

- `org.apache.xmlbeans.`SchemaTypeLoader
- `org.apache.xmlbeans.`SchemaGlobalElement
- `org.apache.xmlbeans.`SchemaType
- `org.apache.xmlbeans.`SchemaProperties

SchemaTypeLoader

The `org.apache.xmlbeans.SchemaTypeLoader` class is used to find specific nodes in the XMLBeans parse tree. In particular, you can use it to find element nodes and type nodes. The most useful methods from the `SchemaTypeLoader` class are shown in Table 17.

Table 17: *Methods from the SchemaTypeLoader Class*

Method Signature	Description

`SchemaGlobalElement findElement(`
`  javax.xml.namespace.QName name`
`)`
	Returns the global element definition with the given name, or null if none.

`SchemaType findType(`
`  javax.xml.namespace.QName name`
`)`
	Returns the type with the given name, or null if none.

SchemaGlobalElement

The `org.apache.xmlbeans.SchemaGlobalElement` class represents an element node in the XMLBeans parse tree. The most useful methods from the `SchemaGlobalElement` class are shown in Table 18.

Table 18: *Methods from the SchemaGlobalElement Class*

Method Signature	Description
`javax.xml.namespace.QName getName()`	Returns the form-unqualified-or-qualified name.
`SchemaType getType()`	Returns the type.

Table 18: *Methods from the SchemaGlobalElement Class*

Method Signature	Description
`java.math.BigInteger getMinOccurs()`	Returns the `minOccurs` value for this particle.
`java.math.BigInteger getMaxOccurs()`	Returns the `maxOccurs` value for this particle, or null if it is unbounded.
`boolean isNillable()`	True if nillable; always false for attributes.
`String getSourceName()`	The name of the source file in which this component was defined (if known).

SchemaType

The `org.apache.xmlbeans.SchemaType` class represents a type node in the XMLBeans parse tree. The most useful methods from the `SchemaType` class are shown in Table 19.

Table 19: *Methods from the SchemaType Class*

Method Signature	Description
`SchemaStringEnumEntry enumEntryForString( String s )`	Returns the string enum entry corresponding to the given enumerated string, or null if there is no match or this type is not a string enumeration.
`StringEnumAbstractBase enumForInt(int i)`	Returns the string enum value corresponding to the given enumerated string, or null if there is no match or this type is not a string enumeration.
`StringEnumAbstractBase enumForString(String s)`	Returns the string enum value corresponding to the given enumerated string, or null if there is no match or this type is not a string enumeration.
`SchemaType[] getAnonymousTypes()`	The array of inner (anonymous) types defined within this type.
`int getAnonymousUnionMemberOrdinal()`	For anonymous types defined inside a union only: gets the integer indicating the declaration order of this type within the outer union type, or zero if this is not applicable.
`SchemaAttributeModel getAttributeModel()`	Returns the attribute model for this complex type (with simple or complex content).

Table 19: *Methods from the SchemaType Class*

Method Signature	Description
`SchemaProperty[] getAttributeProperties()`	Returns all the SchemaProperties corresponding to attributes.

`SchemaProperty getAttributeProperty(`
`  QName attrName`
`)`
	Returns a SchemaProperty corresponding to an attribute within this complex type by looking up the attribute name.

`SchemaType getAttributeType(`
`  QName eltName,`
`  SchemaTypeLoader wildcardTypeLoader`
`)`
	Returns the type of an attribute based on the attribute name and the type system within which (wildcard) names are resolved.

`QName getAttributeTypeAttributeName()`	Returns the attribute qname if this is a attribute type, or null otherwise.
`SchemaType getBaseEnumType()`	If this is a string enumeration, returns the most basic base schema type that this enuemration is based on.
`SchemaType getBaseType()`	Returns base restriction or extension type.
`SchemaType getContentBasedOnType()`	For complex types with simple content returns the base type for this type's content.
`SchemaParticle getContentModel()`	Returns the complex content model for this complex type (with complex content).
`int getContentType()`	Returns `EMPTY_CONTENT`, `SIMPLE_CONTENT`, `ELEMENT_CONTENT`, or `MIXED_CONTENT` for complex types.
`int getDecimalSize()`	For atomic numeric restrictions of decimal only: the numeric size category.
`int getDerivationType()`	Returns an integer for the derivation type, either `DT_EXTENSION`, `DT_RESTRICTION`, `DT_NOT_DERIVED`.
`SchemaProperty[] getDerivedProperties()`	Returns the SchemaProperties defined by this complex type, exclusive of the base type (if any).
`SchemaProperty[] getElementProperties()`	Returns all the SchemaProperties corresponding to elements.

Table 19: *Methods from the SchemaType Class*

Method Signature	Description

`SchemaProperty getElementProperty(`
`  QName eltName`
`)`
	Returns a SchemaProperty corresponding to an element within this complex type by looking up the element name.

`SchemaType getElementType(`
`  QName eltName,`
`  QName xsiType,`
`  SchemaTypeLoader wildcardTypeLoader`
`)`
	Returns the type of a child element based on the element name and an xsi:type attribute (and the type system within which names are resolved).

`XmlAnySimpleType[] getEnumerationValues()`	Returns the array of valid objects from the enumeration facet, null if no enumeration defined.
`SchemaType getListItemType()`	For list types only: get the item type.
`QName getName()`	The name used to describe the type in the schema.
`SchemaType getPrimitiveType()`	For atomic types only: get the primitive type underlying this one.
`SchemaProperty[] getProperties()`	For atomic types only: get the primitive type underlying this one.
`int getSimpleVariety()`	Returns whether the simple type is `ATOMIC`, `UNION`, or `LIST`.
`SchemaStringEnumEntry[] getStringEnumEntries()`	Returns the array of `SchemaStringEnumEntries` for this type: this array includes information about the java constant names used for each string enum entry.
`SchemaTypeSystem getTypeSystem()`	Returns the `SchemaTypeLoader` in which this type was defined.
`SchemaType getUnionCommonBaseType()`	For union types only: get the most specific common base type of the constituent member types.
`SchemaType[] getUnionConstituentTypes()`	For union types only: get the constituent member types.
`SchemaType[] getUnionMemberTypes()`	For union types only: get the shallow member types.
`SchemaType[] getUnionSubTypes()`	For union types only: gets the full tree of member types.

Table 19: *Methods from the SchemaType Class*

Method Signature	Description
`boolean hasAllContent()`	True if the complex content model for this complex type is an `all` group.
`boolean hasAttributeWildcards()`	True if this type permits wildcard attributes.
`boolean hasElementWildcards()`	True if this type permits element wildcards.
`boolean hasPatternFacet()`	True if there are regular expression pattern facets.
`boolean hasStringEnumValues()`	True if this is a string enum where an integer is assigned to each enumerated value.
`boolean isAnonymousType()`	True if the Xsd type is anonymous (i.e., not top-level).
`boolean isAttributeType()`	True if this is a attribute type.
`boolean isBounded()`	True if bounded.
`boolean isBuiltinType()`	True for any of the 40+ built-in types.
`boolean isNoType()`	True for the type object that represents a the absence of a determined type.
`boolean isNumeric()`	True if numeric.
`boolean isPrimitiveType()`	True for any of the 20 primitive types (plus `anySimpleType`).
`boolean isSimpleType()`	True for the anySimpleType and any restrictions/unions/lists.
`boolean isURType()`	True for `anyType` and `anySimpleType`.
`boolean matchPatternFacet(String s)`	True if the given string matches the pattern facets.
`int ordered()`	True if ordered.
`QNameSet qnameSetForWildcardAttributes()`	Returns a `QNameSet` of attributes that may exist in wildcard buchets and are not explicitly defined in this schema type.
`QNameSet qnameSetForWildcardElements()`	Returns a `QNameSet` of elements that may exist in wildcard buchets and are not explicitly defined in this schema type.

SchemaProperties

The `org.apache.xmlbeans.SchemaProperties` class represents a summary of the element definitions that *share the same name* within a complex type definition. Rather than having to look up the properties for all of the different element fields that have the same name, it is usually simpler to obtain the relevant `SchemaProperties` object. The `SchemaProperties` object attempts to unify the properties of the same-name elements in a consistent manner.

The most useful methods from the `SchemaProperties` class are shown in Table 19.

Table 20: *Methods from the SchemaProperties Class*

Method Signature	Description
`SchemaType getContainerType()`	The type within which this property appears.
`String getDefaultText()`	Returns the default or fixed value, if it is consistent.
`XmlAnySimpleType getDefaultValue()`	Returns the default or fixed value as a strongly-typed value, if it is consistent.
`BigInteger getMaxOccurs()`	Returns a summarized maximum occurrence number.
`BigInteger getMinOccurs()`	Returns a summarized minimum occurrence number.
`QName getName()`	The name of this element or attribute.
`SchemaType getType()`	The schema type for the property.
`int hasDefault()`	Returns `NEVER`, `VARIABLE`, or `CONSISTENTLY` defaulted, depending on the defaults present in the elements in this property.
`int hasFixed()`	Returns `NEVER`, `VARIABLE`, or `CONSISTENTLY` fixed, depending on the fixed constraints present in the elements in this property.
`int hasNillable()`	Returns `NEVER`, `VARIABLE`, or `CONSISTENTLY` nillable, depending on the nillability of the elements in this property.
`boolean isAttribute()`	True for attributes.
`boolean isReadOnly()`	True for read-only properties.

APPENDIX A

Java Utility Classes

For you convenience, this appendix summarizes some standard Java utility classes that are used extensively throughout the WSDLGen scripts.

In this appendix

This appendix discusses the following topics:

Useful Java Utility Classes

Overview

There are a few Java utility classes that are extensively used in the WSDLGen scripts, as follows:

- `javax.xml.namespace.QName`
- `java.util.Map`
- `java.util.Collection`
- `java.util.Iterator`
- `java.util.List`
- `java.util.ListIterator`

For your convenience, the API for these utility classes is summarized here. This summary does not include all of the methods in these classes, however. For the complete Java API, consult the Javadoc reference on Sun's Web site:

http://java.sun.com/j2se/1.5.0/docs/api/

javax.xml.namespace.QName

The `javax.xml.namespace.QName` class includes the methods shown in Table 21.

Table 21: *Some Methods and Constructors from QName*

Method/Constructor Signature	Description
`QName(String localPart)`	Construct a QName that has no namespace.
`QName(String namespaceURI, String localPart)`	Construct a QName consisting of a namespace URI and a local part.
`QName(String namespaceURI, String localPart, String Prefix)`	Constructor with namespace prefix (the prefix is not very important in the context of WSDL parsing).
`String getLocalPart()`	Get the local part of the QName.
`String getNamespaceURI`	Get the namespace URI of the QName.
`String getPrefix`	Get the prefix (rarely needed).
`String toString()`	Return `"{"+namespaceURI+"}"+localPart`.

java.util.Map

The `java.util.Map<K,V>` class includes the methods shown in Table 22.

Table 22: *Some Methods from java.util.Map*

Method Signature	Description
`put(K key, V value)`	Add a new entry to the map.
`V get(Object key)`	Use the key to look up a value in the map.
`java.util.Collection<V> values()`	If you want to iterate over all of the values in the map, it is necessary to convert it to a collection first.
`boolean isEmpty()`	True, if the map is empty.
`int size()`	Return the number of entries in the map.

java.util.Collection

The `java.util.Collection<E>` class includes the methods shown in Table 23.

Table 23: *Some Methods from java.util.Collection*

Method Signature	Description
`java.util.Iterator<E> iterator()`	Return an iterator, which can be used to iterate over all members of the collection.

java.util.Iterator

The `java.util.Iterator<E>` class includes the methods shown in Table 24.

Table 24: *Some Methods from java.util.Iterator*

Method Signature	Description
`boolean hasNext()`	True, if a call to `next()` would return another element in the collection.
`E next()`	Return the next element in the collection and increment the iterator index.

java.util.List

The `java.util.List<E>` class includes the methods shown in Table 25.

Table 25: *Some Methods from java.util.List*

Method Signature	Description
`Object[] toArray()`	Convert the list to an array.
`java.util.ListIterator listIterator()`	Return an iterator, which you can use to iterate over all of the list members.
`boolean isEmpty()`	True, if the list is empty.
`int size()`	Return the number of list members.

java.util.ListIterator

The `java.util.ListIterator<E>` class, which is a bidirectional iterator, includes the methods shown in Table 26.

Table 26: *Some Methods from java.util.ListIterator*

Method Signature	Description
`boolean hasNext()`	True, if a call to `next()` would return another list member.
`E next()`	Return the next member of the list and increment the iterator index.
`boolean hasPrevious()`	True, if a call to `previous()` would return another list member.
`E previous()`	Return the previous member of the list and decrement the iterator index.

Index

V

W

X