Progress | Artix

Router Guide, C++ Runtime

Version 5.6, December 2011

BUSINESS MAKING PROGRESS™ PROGRESS software

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

List of Tables

Preface

What is Covered in this Book

This book discusses how to use the Artix ESB for C++ routing service. It covers how the routing service directs message, the WSDL extensions used to define routing rules, and how to deploy an instance of the routing service.

Who Should Read this Book

This book is intended for any user who needs to use the Artix routing service to connect endpoints in a SOA. It is expected that the reader have a basic understanding of Service Oriented design concepts and WSDL.

How to Use this Book

For an overview of the routing service, read Chapter 1, “Introduction.”

For information on writing routing rules, read:

- Chapter 2, “Compatibility of Ports and Operations.”
- Chapter 3, “Creating a Basic Route.”
- Chapter 4, “Adding Operation-Based Rules to a Route.”
- Chapter 5, “Adding Attribute-Based Rules to a Route.”
- Chapter 6, “Adding Content-Based Rules to a Route.”
- Chapter 8, “Linking Routes.”
- Chapter 9, “Creating Routes Using Artix Tools.”

For information on configuring the routing service and optimizing its performance, read:

- Chapter 10, “Deploying an Artix Router.”
- Chapter 11, “Routing Messages Containing References.”

For information on the advanced features of the router, read Chapter 7, “Using Advanced Routing Features.”

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

Introduction

The Artix routing service provides message routing based on operations, ports, message attributes, or message content.

In this chapter

This chapter discusses the following topics:

Features of the Routing Service

Overview

An Artix router redirects messages based on rules defined in an Artix contract. The routing functionality is provided by an Artix plug-in and configuration. This means that neither the client nor the server endpoints need to be modified, nor are they are aware that routing is occurring. An Artix router is sometimes referred to as an Artix *switch*.

Routes

The most basic Artix routes are between two endpoints that are described by the `port` element of a WSDL contract. You can refine your routes using the following types of additional rules:

- Operation-based
- Attribute-based
- Content-based

Operation-based

Operation-based rules allow you to refine a route by specifying a particular operation on which the router will filter messages. By adding an operation-based rule to a route, you direct the router to only act upon messages that originate due to an invocation on a particular operation of the specified port. Messages are routed between logical operations whose arguments are equivalent.

For more information see "Adding Operation-Based Rules to a Route" on page 41.

Attribute-based

Attribute-based routing rules allow you to refine a routing by specifying values in the message header to be inspected. By adding attribute-based rules to a route, you can direct the router to only redirect messages based on certain values specified in the message header.

For more information see "Adding Attribute-Based Rules to a Route" on page 45.

Content-based

Content-based routing rules allow to refine a route by inspecting the contents a message. Adding a content-based rule lets you route messages based on the value of particular elements of a message. The routes are defined using simple XPATH expressions that query the message content and select a destination based on the result.

For more information see “Adding Content-Based Rules to a Route” on page 49.

Advanced features

In addition, you can specify routes that give you the following advanced capabilities:

- Failover
- Load balancing
- Message broadcasting (fanout)

For more information see “Using Advanced Routing Features” on page 59.

Routing Contracts

Overview

A router's contract must include definitions for the source services and destination services. The contract also defines the routes that connect the source endpoints to the destination endpoints. These routing rules is all that is required to implement a route.

Routing contract requirements

A contract for the routing service is very similar to a contract for any other Artix service. It is a WSDL document that defines the types, interfaces, data mappings, and networking information that defines an endpoint. Because the routing service bridges two, or more endpoints, it requires that all of the information for the endpoints it bridges are defined. In addition, a routing service contract contains information specifying the routing rules for connecting the defined endpoints.

A contract for the routing service must specify the following:

- all of the types passed between all of the endpoints being connected.
- all of the messages that can be passed between the endpoints being connected.
- an interface definition for each of the endpoints being connected.

  Note: A routing service contract may have only one interface definition because multiple endpoints can share the same interface.

- a binding definition for each endpoint being connected.
- the connection information for all of the endpoints being connected.
- at least one set of routing rules to define how messages are routed between the connected endpoints.

Routing namespace

The WSDL extension used to specify routes in an Artix contract are defined in the namespace `http://schemas.iona.com/routing`. When describing routes in an Artix contract you must add the following to your contract's definition element:

```
<definitions ...
  xmlns:routing="http://schemas.iona.com/routing"
  ...>
```

Common routing extensions

The most commonly used of the routing extensions are:

routing:route is the root element of any route defined in the contract.

routing:source specifies the port that acts as the source for messages that are to be routed.

routing:destination specifies the port to which messages will be routed.

You do not need to do any programming and your applications need not be aware that any routing is taking place.

Router Deployment Patterns

Overview

An Artix router does not require that any Artix-specific code be compiled or linked into existing applications. An Artix router is created by loading the Artix `routing` plug-in into an Artix process. The recommended way to deploy a router is to use the Artix container (see *Deploying Artix Solutions*).

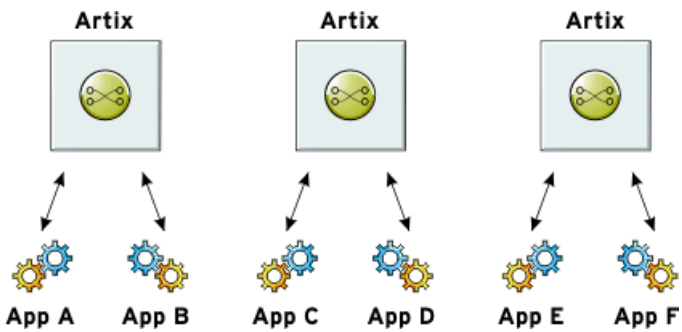
Artix router can be deployed in a number of ways. Two common deployment patterns are:

- Deploying multiple routers—each bridging between two applications.
- Deploying one router—it bridges between all applications in a domain.

Deploying multiple routers

This approach simplifies designing integration solutions, and provides faster processing of each message (shown in Figure 1). Using this approach, the Artix contract describing the interaction of the applications is simpler. It contains only the logical interfaces shared by the two applications, the bindings for each payload format, and the routing rules.

Figure 1: *Using Multiple Artix Routers for Single Routes*

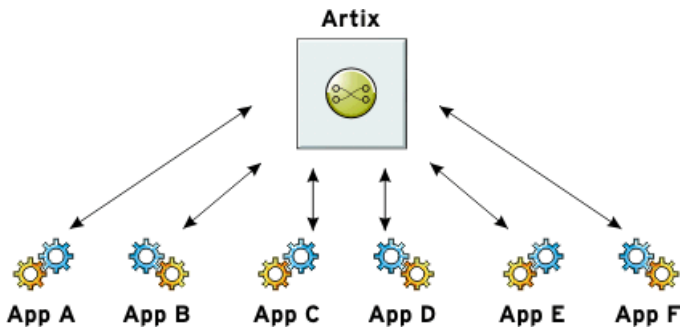


Because most applications use only one network transport, the number of ports is minimal and the routing rules are simple. Keeping the contract simple also enhances the performance of each router because it has less processing to do. In this approach, each router's resource usage can be limited by tailoring its configuration to optimize the router for the integration task that it is responsible for.

Deploying one router

This approach limits the number of external services required in your deployment environment (shown in Figure 2). This can simplify monitoring and installation of deployments. It also reduces the number of moving parts in an integration solution.

Figure 2: *Using a Single Artix Router for Multiple Routes*



Using this approach, you can use a single WSDL contract that includes all the information for all routes. In this case, the contract information that describes the interaction of the applications is more complex. It contains the logical interfaces shared by multiple applications, the bindings for each payload format, and the routing rules.

Alternatively, you can also specify that a single router uses multiple WSDL files, each of which describes a single route, or a number of routes. These could be the same WSDL contracts used in multiple router deployment, however, they are all deployed in the same router process. The configuration that identifies the WSDL file containing the routing details is specified using a list, which can include a collection of multiple WSDL files. For more information, see "Configuring an Artix Router" on page 76.

CHAPTER 2

Compatibility of Ports and Operations

The source endpoint and destination endpoint of a route must be able to consume the routed messages.

Overview

The routing service can route messages between endpoints that expect similar messages. The endpoints can use different message transports and different payload formats, but the messages must be logically identical. For example, if you have a baseball scoring service that is hosted on a mainframe, it might send data using fixed record length fields over a WebSphere MQ queue. Using a router, you can route the score data to a reporting service that consumes SOAP messages over HTTP.

Using the most basic routing rules, the destination endpoint must have a matching logical operation defined for each of the logical operations defined for the source endpoint. If you add an operation-based rule, the restriction on the endpoints is relaxed. The source endpoint and the destination endpoint must have one logical operation that uses messages with the same logical description.

Routing between endpoints

Routing between endpoints is rough grained in that the routing rules are defined on the `port` elements of an Artix contract and do not look at the individual logical operations defined in the logical interface, defined by a `portType` element, for which the `port` element defines an endpoint. Therefore, basic routing rules require that the endpoints between which messages are routed must have compatible logical interface descriptions.

For two endpoints to have compatible logical interfaces the following conditions must be met:

- The `portType` element defining the destination's logical interface must contain a matching `operation` element for each `operation` element in the `portType` element defining the source's logical interface. Matching `operation` elements must have the same value in their `name` attribute.
- Each of the matching `operation` elements must have the same number of `input`, `output`, and `fault` elements.
- Each of the matching `operation` elements' `input` elements must be associated to a logical message, defined by a `message` element, whose sequence of part `elements` have matching types.
- Each of the matching `operation` elements' `output` elements must be associated to a logical message whose sequence of part `elements` have matching types.
- Each of the matching `operation` elements' `fault` elements must be associated to a logical message whose sequence of part `elements` have matching types.

For example, given the two logical interfaces defined in Example 1 you could construct a route from an endpoint bound to `baseballScorePortType` to an endpoint bound to `baseballGamePortType`. However, you could not create a route from an endpoint bound to `finalScorePortType` to an endpoint bound to `baseballGamePortType` because the message types used for the `getScore` operation do not match.

Example 1: *Logical interface compatibility example*

```
<message name="scoreRequest>
  <part name="gameNumber" type="xsd:int"/>
</message>
<message name="baseballScore">
  <part name="homeTeam" type="xsd:int"/>
  <part name="awayTeam" type="xsd:int"/>
  <part name="final" type="xsd:boolean"/>
</message>
<message name="finalScore">
  <part name="home" type="xsd:int"/>
  <part name="away" type="xsd:int"/>
  <part name="winningTeam" type="xsd:string"/>
</message>
<message name="winner">
  <part name="winningTeam" type="xsd:string"/>
</message>
<portType name="baseballGamePortType">
  <operation name="getScore">
    <input message="tns:scoreRequest" name="scoreRequest"/>
    <output message="tns:basballScore" name="baseballScore"/>
  </operation>
  <operation name="getWinner">
    <input message="tns:scoreRequest" name="winnerRequest"/>
    <output message="tns:winner" name="winner"/>
  </operation>
</portType>
<portType name="baseballScorePortType">
  <operation name="getScore">
    <input message="tns:scoreRequest" name="scoreRequest"/>
    <output message="tns:basballScore" name="baseballScore"/>
  </operation>
</portType>
<portType name="finalScorePortType">
  <operation name="getScore">
    <input message="tns:scoreRequest" name="scoreRequest"/>
    <output message="tns:finalScore" name="finalScore"/>
  </operation>
</portType>
```

Routing between operations

Operation-based routing rules check for compatibility based on the `operation` elements of an endpoint's logical interface description. Therefore, messages can be routed between any two compatible logical operations.

The following conditions must be met for operations to be compatible:

- The operations must have the same number of `input`, `output`, and `fault` elements.
- The logical messages must have the same sequence of part types.

For example, if you added the logical interface in Example 2 to the interfaces in Example 1 on page 35, you could specify a route from `getFinalScore` defined in `fullScorePortType` to `getScore` defined in `finalScorePortType`. You could also define a route from `getScore` defined in `fullScorePortType` to `getScore` defined in `baseballScorePortType`.

Example 2: *Operation-based routing interface*

```
<portType name="fullScorePortType">
  <operation name="getScore">
    <input message="tns:scoreRequest" name="scoreRequest"/>
    <output message="tns:basballScore" name="baseballScore"/>
  </operation>
  <operation name="getFinalScore">
    <input message="tns:scoreRequest" name="scoreRequest"/>
    <output message="tns:finalScore" name="finalScore"/>
  </operation>
</portType>
```

CHAPTER 3

Creating a Basic Route

The simplest route directs messages between two endpoints without any conditions.

Overview

Basic routing rules simply specify the source endpoint, or endpoints, for the messages and the destination endpoint to which messages are routed. All messages received by the source endpoint are routed to the destination endpoint.

To describe a basic routing rule you use three elements:

- routing:route
- routing:source
- routing:destination

---

routing:route

The `routing:route` element is the root element of each route you describe in your contract. It takes one required attribute, `name`, that specifies a unique identifier for the route. The `routing:route` element also has an optional attribute, `multiRoute`, which is discussed in "Using Advanced Routing Features" on page 59.

---

routing:source

The `routing:source` element specifies the endpoint on which the route listens for messages. A route can have several `routing:source` elements as long as they all meet the compatibility rules discussed in "Routing between endpoints" on page 34.

The `routing:source` element requires two attributes described in Table 1.

Table 1: *Required Attributes for routing:source*

Attribute	Description
`service`	Specifies the name of the `service` element in which the source endpoint is defined.
`port`	Specifies the name of the `port` element defining the source endpoint.

routing:destination

The `routing:destination` element specifies the endpoint to which the source messages are routed. The destination endpoint must be compatible with the source endpoint. For a discussion of the compatibility rules see "Routing between endpoints" on page 34.

In standard routing only one destination is allowed per route. Multiple destinations are allowed in conjunction with the `routing:route` element's `multiRoute` attribute that is discussed in "Using Advanced Routing Features" on page 59.

The `routing:destination` element requires two attributes described in Table 2..

Table 2: *Required Attributes for routing:destination*

Attribute	Description
`service`	Specifies the name of the `service` element in which the destination endpoint is defined.
`port`	Specifies the name of the `port` element defining the destination endpoint.

Example

For example, to define a route from `baseballScorePortType` to `baseballGamePortType`, defined in Example 1 on page 35, your Artix contract would contain the elements in Example 3.

Example 3: *Port-based routing example*

```
1  <service name="baseballScoreService">
     <port binding="tns:baseballScoreBinding"
           name="baseballScorePort">
       <soap:address location="http://localhost:8991"/>
     </port>
   </service>
   <service name="baseballGameService">
     <port binding="tns:baseballGameBinding"
           name="baseballGamePort">
       <tibrv:port serverSubject="com.mycompany.baseball"/>
     </port>
   </service>
2  <routing:route name="baseballRoute">
     <routing:source service="tns:baseballScoreService"
                     port="tns:baseballScorePort"/>
     <routing:destination service="tns:baseballGameService"
                          port="tns:baseballGamePort"/>
   </routing:route>
```

There are two sections to the contract fragment shown in Example 3:

1. The logical interfaces must be bound to physical ports in `service` elements of the Artix contract.
2. The route, `baseballRoute`, is defined with the appropriate `service` and `port` attributes.

CHAPTER 4

Adding Operation-Based Rules to a Route

Operation-based rules narrow the scope used to define the source of the messages to a specific operation.

Overview

Operation-based routing rules refine a route by narrowing the source of routed messages to specific logical operation. Any message not related to the specified logical operation will be unaffected by the route.

Adding an operation-based rule

To specify an operation-based routing rule you need to specify one additional element to your route description: `routing:operation`. The `routing:operation` element takes one required attribute, `name`, that specifies the value of the `name` attribute of an `operation` element in the source endpoint's logical interface. The specified `operation` element becomes the source of messages that are routed. Messages corresponding to other logical operations will not be routed.

The `routing:operation` element also has one optional attribute, `target`, that specifies the value of the `name` attribute of an `operation` element in the destination endpoint's logical interface. The specified `operation` element becomes the destination of messages redirected by the route. If a target is specified, messages are routed between the two operations. If no target is specified, the source operation's name is used as the name of the target operation. The source and target operations must meet the compatibility requirements discussed in "Routing between operations" on page 36.

You can specify any number of `routing:operation` elements in a route. They must be specified after all of the `routing:source` elements and before any `routing:destination` elements.

How operation-based rules are applied

Operation-based routing rules apply to all of the `routing:source` elements in the route. Therefore, if an operation-based routing rule is specified, a message will be routed if all of the following are true:

- The message is received from one of the endpoints specified in a `routing:source` element.
- The operation name associated with the received message is specified in one of the `routing:operation` elements.

If there are multiple operation-based rules in the route, the message will be routed to the destination specified by the first the matching operation's `target` attribute.

Example

For example, to route messages from the `getFinalScore` operation defined in `fullScorePortType`, shown in Example 2 on page 36, to the `getScore` operation defined in `finalScorePortType`, shown in Example 1 on page 35, your Artix contract would contain the elements in Example 4.

Example 4: *Operation to Operation Routing*

```
1  <service name="fullScoreService">
     <port binding="tns:fullScoreBinding"
           name="fullScorePort">
       <mq:server QueueManager="BBQM"
                  QueueName="MLBQueue"
                  ReplyQueueManager="BBRQM"
                  ReplyQueueName="MLBScoreQueue"/>
     </port>
   </service>
   <service name="finalScoreSerice">
     <port binding="tns:finalScoreBinding"
           name="finalScorePort">
       <soap:address location="http://artie.com/finalScoreServer"/>
     </port>
   </service>
2  <routing:route name="scoreRoute">
     <routing:source service="tns:fullScoreService"
                     port="tns:fullScorePort"/>
     <routing:operation name="getFinalScore" target="getScore"/>
     <routing:destination service="tns:finalScoreService"
                          port="tns:finalScorePort"/>
   </routing:route>
```

There are two sections to the contract fragment shown in Example 4:

1. The logical interfaces must be bound to physical endpoints in `service` elements of the Artix contract.
2. The route, `scoreRoute`, is defined using the `routing:operation` element.

You could also create a route between the operation `getScore`, defined in `baseballGamePortType`, and an endpoint bound to `baseballScorePortType`. See Example 1 on page 35.The resulting contract would include the fragment shown in Example 5.

Example 5: *Operation to Port Routing Example*

```
<service name="baseballGameService">
  <port binding="tns:baseballGameBinding"
        name="baseballGamePort">
    <soap:address location="http://localhost:8991"/>
  </port>
</service>
<service name="baseballScoreService">
  <port binding="tns:baseballScoreBinding"
        name="baseballScorePort">
    <iiop:address location="file:\\score.ref"/>
  </port>
</service>
<routing:route name="scoreRoute">
  <routing:source service="tns:baseballGameService"
                  port="tns:baseballGamePort"/>
  <routing:operation name="getScore"/>
  <routing:destination service="tns:baseballScoreService"
                       port="tns:baseballScorePort"/>
</routing:route>
```

Note that the `routing:operation` element only uses the `name` attribute. In this case the logical interface bound to `baseballScorePort`, `baseballScorePortType`, must contain an operation `getScore` that has matching messages as discussed in "Routing between operations" on page 36.

CHAPTER 5

Adding Attribute-Based Rules to a Route

Attribute-based rules refine a route by selecting the messages to be routed based on the transport attributes set in a message's header.

Overview

Artix allows you to route messages based on the transport attributes set in a message's header when using HTTP or WebSphere MQ. You can also route messages based on security settings and the CORBA principle.

Adding attribute-based rules

Rules the select messages based on message header transport attributes are defined in `routing:transportAttribute` elements in the route definition. Transport attribute rules are defined after all of the operation-based routing rules and before any destinations are listed.

The criteria for determining if a message meets an attribute-based rule are specified in sub-elements of the `routing:tranportAttribute` element. A message passes the rule if it meets each criterion specified in the listed sub-element.

Defining the attributes

Each sub-element requires the two attributes defined in Table 3.

Table 3: *Required Attributes for Attribute Selection Elements*

Attribute	Description
`contextName`	Specifies the context defining the transport attribute being evaluated.
`contextAttributeName`	Specifies the name of the transport attribute being evaluated.

The `contextName` attribute is specified using the QName of the context in which the attribute is defined. The contexts shipped with Artix are described in Table 4.The `contextAttributeName` is also a QName and is relative to the context specified. For example, `UserName` is a valid attribute name for any of the HTTP contexts, but not for the MQ contexts.

Table 4: *Context QNames*

Context QName	Details
`http-conf:HTTPServerIncomingContexts`	Contains the attributes for HTTP messages being received by a service.
`corba:corba_input_attributes`	Contains the data stored in the CORBA principle.
`mq:IncomingMessageAttributes`	Contains the attributes for MQ messages being received by a service.
`bus-security`	Contains the attributes used by the Artix security service to secure your services.

Most sub-elements have a `value` attribute that can be tested. When dealing with string comparisons all elements have an optional `ignorecase` attribute that can have the values `yes` or `no` (`no` is the default). Each of the sub-elements can occur zero or more times, in any order:

routing:equals applies to string or numeric attributes. For strings, the `ignorecase` attribute may be used.

routing:greater applies only to numeric attributes and tests whether the attribute is greater than the value.

routing:less applies only to numeric attributes and tests whether the attribute is less than the value.

routing:startswith applies to string attributes and tests whether the attribute starts with the specified value.

routing:endswith applies to string attributes and tests whether the attribute ends with the specified value.

routing:contains applies to string or list attributes. For strings, it tests whether the attribute contains the value. For lists, it tests whether the value is a member of the list. The `contains` element accepts the optional `ignorecase` attribute for both strings and lists.

routing:empty applies to string or list attributes. For lists, it tests whether the list is empty. For strings, it tests for an empty string.

routing:nonempty applies to string or list attributes. For lists, it passes if the list is not empty. For strings, it passes if the string is not empty.

For information on the transport attributes for HTTP and WebSphere MQ see Binding and Transports, C++ Runtime.

Example

Example 6 shows a route using attribute-based rules based on HTTP header attributes. Only messages sent to the server whose `UserName` is equal to `JohnQ` will be passed through to the destination port.

Example 6: *Transport Attribute Rules*

```
<routing:route name="httpTransportRoute">
  <routing:source service="tns:httpService"
                  port="tns:httpPort"/>
  <routing:transportAttributes>
    <routing:equals
               contextName="http-conf:HTTPServerIncomingContexts"
               contextAttributeName="UserName"
               value="JohnQ"/>
  </routing:transportAttributes>
  <routing:destination service="tns:httpDest"
                       port="tns:httpDestPort"/>
</routing:route>
```

CHAPTER 6

Adding Content-Based Rules to a Route

Content-based routing rules evaluate the contents of a message and routes it based on the results.

Procedure

To create a content-based route rule in your contract you need to do the following things:

1. Add an expression to select message content using a `routing:expression` element.
2. Add a new route to you contract using a `routing:route` element.
3. Add a source endpoint to your route using a `routing:source` element.
4. Specify the expression to use as a routing criteria using a `routing:query` element.
5. Add one or more `routing:destination` elements as children to the `routing:query` element.
6. If you want to add a default destination endpoint, add a `routing:destination` element as a child of the `routing:route` element.

In this section

This section discusses the following topics:

Router's Message Representation

Overview

The router receives messages in a number of wire formats. It uses the information provided in the `binding` element of its contract to turn the raw message into an XML message that can be evaluated. Before you can write an expression to select content from a message passing through the router, you need to understand how the router sees the message.

Doc-literal style contracts

If your contract is constructed using the recommended doc-literal style, the router sees the message as an instance of the element specified as the message part. For example, if your service was defined by the WSDL fragment in Example 7, the router would see a message with the root element `ticket`.

Example 7: *Doc-literal WSDL Fragment*

```
<definitions targetNamespace="vehicle.demo.example"
             xmlns:tns="vehicle.demo.example"
             ...>
  <types ...>
  ...
    <complexType name="vehicleType">
      <sequence>
        <element name="vin" type="xsd:string" />
        <element name="model" type="xsd:string" />
      </sequence>
    </complexType>
    <complexType name="ticketType">
      <sequence>
        <element name="vehicle" type="vehicleType" />
        <element name="name" type="xsd:string" />
        <element name="parkTime" type="xsd:string" />
      </sequence>
    </complexType>
    <element name="ticket" type="ticketType" />
    ...
  </types>
...
<message name="ticketRequest">
  <part name="myTicket" element="xsd1:ticket" />
</message>
...
```

Example 7: *Doc-literal WSDL Fragment (Continued)*

```
<portType name="parkingLotMeter">
  <operation name="register">
    <input name="parkedCar" message="tns:ticketRequest"/>
    ...
  </operation>
...
</portType>
...
```

Example 8 shows an example of the message that the router would process given the WSDL in Example 7.

Example 8: *Doc-literal Router Message*

```
<ns1:parkedCar xmlns:ns1="vehicle.demo.example">
  <ticket>
    <vehicle>
      <VIN>0123456789</VIN>
      <model>Prius</model>
    </vehicle>
    <name>Old MacDonald</name>
    <time>19:00</time>
  </ticket>
</ns1:parkedCar>
```

Non-standard contracts

When you use non-standard messages in your contract, the router sees the message as a virtual XML document that is reconstructed from the WSDL definitions in the contract. The mapping is done as follows:

1. The name of the message's root element is the QName of the `message` element referred to by the operation's `input` element.
2. Each `part` element of the message referenced by the `input` element is mapped to an element derived from the `name` attribute of the `part` element.
3. If the `part` element is of a complex type, or an element of a complex type, the type's elements appear inside of the element corresponding to the `part` element.

For example, if you had a service defined by the WSDL fragment in Example 9 and were going to route requests to the register operation, the router would scan an XML document constructed using the message `ticketRequest`, which is the input message.

Example 9: *Non-standard WSDL Fragment*

```
<definitions targetNamespace="vehicle.demo.example"
             xmlns:tns="vehicle.demo.example"
             ...>
  <types ...>
  ...
    <complexType name="vehicleType">
      <element name="vin" type="xsd:string" />
      <element name="model" type="xsd:string" />
    </complexType>
  ...
  </types>
...
<message name="ticketRequest">
  <part name="vehicle" type="xsd1:vehicleType"/>
  <part name="name" type="xsd:string"/>
  <part name="parkTime" type="xsd:string" />
</message>
...
<portType name="parkingLotMeter">
  <operation name="register">
    <input name="parkedCar" message="tns:ticketRequest"/>
    ...
  </operation>
...
</portType>
...
```

When the router reconstructs the message, it the input message's name, given in the `input` element, as the name of the XML document's root element. It uses the message parts and the schema types to recreate the remaining elements in the XML document. The resulting XML document would look like Example 10.

Example 10: *Router Message*

```
<ns1:parkedCar xmlns:ns1="vehicle.demo.example">
  <vehicle>
    <VIN>0123456789</VIN>
    <model>Prius</model>
  </vehicle>
  <name>Old MacDonald</name>
  <time>19:00</time>
</ns1:parkedCar>
```

Using element names

You can configure the transformer to use the element name of the message parts instead of the value of the `part` element's `name` attribute. For more information see Configuring and Deploying Artix Solutions, C++ Runtime.

Specifying Evaluation Expressions

Overview

The router uses expressions to evaluate a message's content and route it. These expressions are written using the XPath grammar.

Writing XPath expressions

XPath is a standard grammar for addressing the parts of an XML document. The Artix router uses XPath expressions to extract the content of a message for evaluation. For example, if you wanted to write an XPath expression to extract the data stored in the `model` element of the XML document in Example 10 you could use the XPath expression `parkedCar\vehicle\model` which translates into select the `model` element whose parent is a `vehicle` element and has a `parkedCar` element as a parent.

You could also use the XPath expression `\\model` which translates into select all of the `model` elements that are a descendent of the root element. If there were multiple `model` elements, the expression would select them all and return a string representing the node set of `model` elements.

For more information on XPath see the specification at http://www.w3.org/TR/xpath or see the tutorial at http://www.w3schools.com/xpath.

Adding expressions to a contract

You add an expression to your contract using a `routing:expression` element. The `routing:expression` element requires the two attributes described in Table 5.

Table 5: *Required Attributes for routing:expression*

Attribute	Description
`name`	Specifies a unique identifier by which the expression is referred to when used in a route definition.
`evaluator`	Specifies the type of expression being used to select the content.

Note: XPath is the only supported grammar and is specified using the string `xpath`.

Example

Example 11 shows an example of adding an expression to an Artix contract.

Example 11: *Expression in an Artix Contract*

```
<routing:expression name="widgetSize" evaluator="xpath">
  /*/widgetOrderform/type
</routing:expression>
```

The expression selects the `type` child element of the `widgetOrderForm` element in the message. The `widgetOrderForm` element is not the root element of the message. It is generated from one of the `part` elements defined in the contract.

Adding a Content-Based Rule to a Route

Using expressions in a route

To use the expression to route messages, you need to add it to the route. This is done using the `routing:query` element. The `routing:query` element is a child of the `routing:route` element and must follow a single `routing:source` element. It has one attribute, `expression`, that specifies the name of the expression used to select a destination endpoint.

Specifying destinations for a content based routing rule

The destinations that can be selected by the expression are specified using `routing:destination` elements that are children of the `routing:query` element. When used in content-based routing rules, the `routing:destination` elements use the `value` attribute. The `value` attribute specifies the value of the expression that will select the destination endpoint.

For example, the route shown in Example 12 specifies a content-based routing rule that uses the expressing defined in Example 11 and has three possible destination endpoints.

Example 12: *Content-Based Routing Rule*

```
<routing:route name="sizeRoute">
  <routing:source service="tns:orderService" />
  <routing:query expression="tns:widgetSize">
    <routing:destination value="small"
                         service="tns:smallService" />
    <routing:destination value="med" service="tns:medService" />
    <routing:destination value="big" service="tns:bigService" />
  </routing:query>
</routing:route>
```

If the value of the message's `type` element is `med`, the message will be routed to the endpoint defined by the contract's `service` element whose `name` attribute equals `medService`.

Adding a default destination

To add a default destination for a content based routing rule, you simply add a `routing:destination` element after the `routing:query` element. If none of the destination endpoints specified by the content-based routing rule are selected, the first destination after the `routing:query` element is selected. Example 13 shows a content-based routing rule with a default destination endpoint.

Example 13: *Content-Based Routing Rule with a Default Destination*

```
<routing:route name="sizeRoute">
  <routing:source service="tns:orderService" />
  <routing:query expression="tns:widgetSize">
    <routing:destination value="small"
                         service="tns:smallService" />
    <routing:destination value="med" service="tns:medService" />
    <routing:destination value="big" service="tns:bigService" />
  </routing:query>
  <routing:destination service="tns:miscService" />
</routing:route>
```

CHAPTER 7

Using Advanced Routing Features

The router has a number of advanced features that use multiple destinations.

Overview

Artix routing also supports the following advanced routing capabilities:

- Load balancing between a number of endpoints.
- Broadcasting a message to a number of destinations.
- Specifying a failover service to which messages are routed.

All of these features use the optional `multiRoute` attribute on the `rotuing:route` element.

In this chapter

This chapter discusses the following topics:

Load Balancing

Overview

The router can load balance requests across a number of endpoints without requiring any special configuration or programming. It uses a round-robin algorithm to route requests, that match a routing rule, to one of the specified destination endpoints.

Specifying router based load balancing

Router-based load balancing rules are defined using the `routing:route` element's `multiRoute` attribute. To define a failover route you set the `multiRoute` attribute to `loadBalance`. Within the route definition you define a message source as you would for any other route. You also specify a number of destination endpoints to which messages will be routed. Using a round-robin algorithm the router will direct each request from the source endpoint to one of the specified destination endpoints.

Example

For example, if you had three endpoints that could process requests for baseball scores and wanted to balance the request load among them, you could create a route similar to the one shown in Example 14.

Example 14: *Router Based Load Balancing*

```
<routing:route name="scoreRoute" nultiRoute="loadBalance">
  <routing:source service="tns:baseballGameService"
                  port="tns:baseballGamePort"/>
  <routing:operation name="getScore"/>
  <routing:destination service="tns:baseballScoreService1"
                       port="tns:baseballScorePort"/>
  <routing:destination service="tns:baseballScoreService2"
                       port="tns:baseballScorePort"/>
  <routing:destination service="tns:baseballScoreService3"
                       port="tns:baseballScorePort"/>
</routing:route>
```

Using this route, each time a new request was received for the `getScore` operation, the router would direct it to whichever endpoint was next in the rotation. So, the first request would be routed to `baseballScoreService1`, the second request would be routed to `baseballScoreService2`, the third request would be routed `baseballScoreService3`, and so forth.

Message Broadcasting

Overview

Using the router, you can broadcast a message to multiple endpoints. For example, you could deploy an endpoint whose function is to generate shutdown warnings to all services deployed in a network. You could simplify the development of this service by using an Artix router to intercept a single warning message and broadcast it to the other services. In this way, you would only need to change the router's contract when you add or remove services.

Defining broadcasting rules

You define rules by setting the `multiRoute` attribute in the `routing:route` element to `fanout` in your route definition. This causes routed messages to be transmitted to all of the endpoints specified by the route's `routing:destination` elements.

There are three restrictions to using the fanout method of message broadcasting:

- All of the source endpoints and destination endpoints must be oneways. In other words, they cannot have any output messages.
- The source endpoints and destination endpoints cannot have any fault messages.
- The input messages of the source endpoints and destination endpoints must meet the compatibility requirements as described in "Compatibility of Ports and Operations" on page 33.

Example

Example 15 shows an Artix contract fragment describing a route for broadcasting a message to a number of endpoints.

Example 15: *Fanout Broadcasting*

```
<message name="statusAlert">
  <part name="alertType" type="xsd:int"/>
  <part name="alertText" type="xsd:string"/>
</message>
<portType name="statusGenerator">
  <operation name="eventHappens">
    <input message="tns:statusAlert" name="statusAlert"/>
  </operation>
</portType>
```

Example 15: *Fanout Broadcasting (Continued)*

```
<portType name="statusChecker">
  <operation name="eventChecker">
    <input message="tns:statusAlert" name="statusAlert"/>
  </operation>
</portType>
<service name="statusGeneratorService">
  <port binding="tns:statusGeneratorBinding"
        name="statusGeneratorPort">
    <soap:address location="http:\\localhost:8081"/>
  </port>
</service>
<service name="statusCheckerService">
  <port binding="tns:statusCheckerBinding"
        name="statusCheckerPort1">
    <corba:address location="file:\\status1.ref"/>
  </port>
  <port binding="tns:statusCheckerBinding"
        name="statusCheckerPort2">
    <tuxedo:server>
      <tuxedo:service name="personalInfoService">
        <tuxedo:input operation="infoRequest"/>
      </tuxedo:service>
    </tuxedo:server>
  </port>
</service>
<routing:route name="statusBroadcast" multiRoute="fanout">
  <routing:source service="tns:statusGeneratorService"
                  port="tns:statusGeneratorPort"/>
  <routing:operation name="eventHappens" target="eventChecker"/>
  <routing:destination service="tns:statusCheckerService"
                       port="tns:statusCheckerPort1"/>
  <routing:destination service="tns:statusCheckerService"
                       port="tns:statusCheckerPort2"/>
</routing:route>
```

Failover Routing

Overview

The Artix router can provide a basic level of high-availability by allowing you to create routes that define failover scenarios. The router will automatically redirect messages to a new endpoint if the current destination fails. The router will attempt to send a request to all the destinations in a route before throwing an exception back to the client.

Defining the failover rules

To define a failover route you set the `routing:route` element's `multiRoute` attribute to `failover`. When you designate a route as failover, the routed message's target is selected using a round-robin algorithm. If the first target in the list is unable to receive the message, it is routed to the second target. The route will traverse the destination list until either one of the target services can receive the message or the end of the list is reached. On the next failure, the router will start searching from the last position on the list. So if the message was routed to the second entry on the list to deal with an initial failure, the router will start directing requests to the third entry on the list to handle the second failure. When the end of the list is reached, the router will start at the beginning again. If the router is unsuccessful in delivering a message after trying each service in the failover route once, the router will report that the message is undeliverable.

Example

Given the route shown in Example 16, the message will first be routed to `destinationPortA`. If service on `destinationPortA` cannot receive the message, it is routed to `destinationPortB`.

Example 16: *Failover Route*

```
<routing:route name="failoverRoute" multiRoute="failover">
  <routing:source service="tns:sourceService"
                  port="tns:sourcePort"/>
  <routing:destination service="tns:destinationServiceA"
                       port="tns:destinationPortA"/>
  <routing:destination service="tns:destinationServiceB"
                       port="tns:destinationPortB"/>
  <routing:destination service="tns:destinationServiceC"
                       port="tns:destinationPortC"/>
</routing:route>
```

If `destinationPortB` fails at some future point, the messages are then routed to `destinationPortC`. If `destinationPortC` cannot receive messages, the router will then try `destinationPortA`. If `destinationPortA` is not available, the router will try `destinationPortB`. If `destinationPortB` is unavailable, the router will report that the message cannot be delivered.

Linking Routes

It is possible to create complex routes by linking together several types of routes.

Overview

There are occasions, particularly when using content-based routing or using one of the multi-endpoint routing features, when you need to link together a number of routing criteria. Using the routing service you can do this by linking together a number of routes. For example, you may want to route orders for customers in Brazil to a local endpoint, but you also want the orders to automatically fail-over to a alternative endpoint. You can do this by creating a content-based route that specifies a fail-over route as a destination.

Specifying a route as a destination

You link routes together by specifying one route as the destination of another route. When the destination specifying the linked route is selected, the message is passed through the second route to determine its destination. The second route may also contain destinations that contain linked routes. The message will pass through each linked route in order until a destination containing an endpoint is selected.

To specify a linked route as a destination you replace the `service` attribute and the `port` attribute in a `routing:destination` element with the `route` attribute. The value of the `route` attribute must correspond to the name of another route in the contract. The specified route becomes linked with the destination and any message that selects this destination will be processed through it.

Example

Imagine that your company had order processing centers in several cities and you needed to route orders to the processing center closest to the delivery address. You could implement this using a content-based route as shown in Example 17.

Example 17: *Content-Based Route*

```
<routing:expression name="zipCode" evaluator="xpath">
  tns:placeWidgetOrder/widgetOrderForm/shippingAddress/zipCode
</routing:expression>
<routing:route name="zipCodeRoute">
  <routing:source service="tns:widgetOrderService"
                  port="tns:SOAPPort" />
  <routing:query expression="tns:zipCode">
    <routing:destination value="02452"
                         service="tns:widgetOrderServiceEast"
                         port="walthamPort" />
    <routing:destination value="91105"
                         service="tns:widgetOrderServiceWest"
                         port="passadenaPort" />
  </routing:query>
</routing:route>
```

If you needed to add a fail-over mechanism to ensure that the orders were processed by a different processing center in the event of a failure, you could simply add two linked routes for the destination of the content-based route as shown in Example 18.

Example 18: *Linked Routes*

```
<routing:expression name="zipCode" evaluator="xpath">
  tns:placeWidgetOrder/widgetOrderForm/shippingAddress/zipCode
</routing:expression>
<routing:route name="walthamRoute" multiRoute="failover">
  <routing:destination service="tns:widgetOrderServiceEast"
                       port="walthamPort" />
  <routing:destination service="tns:widgetOrderServiceWest"
                       port="passadenaPort" />
</routing:route>
```

Example 18: *Linked Routes*

```
<routing:route name="passadenaRoute" multiRoute="failover">
  <routing:destination service="tns:widgetOrderServiceWest"
                       port="passadenaPort" />
  <routing:destination service="tns:widgetOrderServiceEast"
                       port="walthamPort" />
</routing:route>
<routing:route name="zipCodeRoute">
  <routing:source service="tns:widgetOrderService"
                  port="tns:SOAPPort" />
  <routing:query expression="tns:zipCode">
    <routing:destination value="02452"
                         route="tns:walthamRoute" />
    <routing:destination value="91105"
                         route="tns:passadenaRoute" />
  </routing:query>
</routing:route>
```

Example 18 expands on Example 17 by adding two routes: `walthamRoute` and `passadenaRoute`. Both of these routes will not perform any routing on their own because they lack `routing:source` elements. They are instead used as destinations for the content-based route called `zipCodeRoute`. In Example 17, the content-based route simply routed to one endpoint for each destination. In Example 18, the route's destinations are linked routes. If the first destination is selected, the message is routed through the fail-over route `walthamRoute`. If the second destination is selected, the message is routed through the fail-over route `passadenaRoute`.

CHAPTER 9

Creating Routes Using Artix Tools

Artix provides both GUI and command-line tools for creating routes.

In this chapter

This chapter discusses the following topics:

Creating Routes from the Command Line

Overview

The `wsdltorouting` command line tool can be used to add routes to contracts. `Wsdltorouting` will import an existing contract and generate a new contract containing the specified routing instructions. The imported contract must contain the specified source endpoint and destination endpoint, otherwise the tool will generate an error.

Usage

To generate a route using the command line tool, use the following command.

```
wsdltorouting [-rn name][-ssn service][-spn port]
              [-dsn service][-dpn port][-on operation]
              [-ta attribute] [-d dir][-o file]
              [-L file][-quiet][verbose][-h][-v] wsdlurl
```

`wsdltorouting` has the following options.

`-rn` *`name`*	Specifies the name of the generated route. If no name is given a unique name will be generated for the route.
`-ssn` *`service`*	Specifies the name of the `service` element to use as the source of the route.
`-spn` *`port`*	Specifies the name of the `port` element to use as the source of the route.
`-dsn` *`service`*	Specifies the name of the `service` element to use as the destination of the route.
`-dpn` *`port`*	Specifies the name of the `port` element to use as the destination of the route.
`-on` *`operation`*	Specifies the name of the operation to use for the route. If the route is port-based, you do not need to use this flag.
`-ta` *`attribute`*	Specifies a transport attribute to use in defining the route. For details on how to specify the transport attributes, see "Specifying transport attributes" on page 71.
`-d` *`dir`*	Specifies the output directory for the generated contract.
`-o` *`file`*	Specifies the filename of the generated contract.
`-L` *`file`*	Specifies the location of your Artix license file. The default behavior is to check `IT_PRODUCT_DIR\etc\license.txt`.

`-quiet`	Specifies that the tool runs in quiet mode.
`-verbose`	Specifies that the tool runs in verbose mode.
`-h`	Displays the tool's usage statement.
`-v`	Displays the tool's version.

Specifying transport attributes

When using `wsdltorouting`, transport attributes are specified using four comma-separated values. The first value specifies the name of the attribute's context. The second value specifies the name of the attribute. The third value is the condition used to evaluate the attribute. The fourth value is the values against which the attribute is evaluated.

Table 6 shows the valid context names to use in specifying a transport attribute.

Table 6: *Context Names Used with wsdltorouting*

Context Name	Artix Context
`HTTP_SERVER_INCOMING_CONTEXTS`	HTTP properties received as part of a client request
`CORBA_CONTEXT_ATTRIBUTES`	CORBA transport properties
`SECURITY_SERVER_CONTEXT`	Properties used to configure security settings

For more information on the properties available in the contexts see either Developing Artix Applications in C++.

Table 7 shows the valid condition entries used in specifying transport attributes when using `wsdltorouting`.

Table 7: *Conditions Used with wsdltorouting*

Condition	WSDL Equivalent
`equals`	routing:equals
`startswith`	routing:startswith
`endswith`	routing:endswith
`contains`	routing:contains

Table 7: *Conditions Used with wsdltorouting (Continued)*

Condition	WSDL Equivalent
`empty`	routing:empty
`nonempty`	routing:nonempty
`greater`	routing:greater
`less`	routing:less

Example

If you had a contract that contained the services `itchy` and `scratchy`, both with an equivalent operation `gouge`, you could use the command shown in Example 19 to add a route to your contract.

Example 19: *Adding a Route with wsdltorouting*

```
wsdltorouting -rn itchyGougeScratchy -ssn itchy -spn gougerPort
        -dsn scratchy -dpn gougedPort -on gouge
       -ta HTTP_SERVER_INCOMING_CONTEXTS,UserName,equals,Goering
        itchyscratchy.wsdl
```

The resulting route is shown in Example 20.

Example 20: *Route from wsdltorouting*

```
<routing:route name="itchyGougeScratchy">
  <routing:source service="tns:itchy"
                  port="tns:gougerPort"/>
  <routing:operation name="gouge"/>
  <routing:transportAttributes>
    <routing:equals
              contextName="http-conf:HTTPServerIncomingContexts"
              contextAttributeName="UserName"
              value="Goering"/>
  </routing:transportAttributes>
  <routing:destination service="tns:scratchy"
                       port="gougedPort"/>
</routing:route>
```

CHAPTER 10

Deploying an Artix Router

An instance of the Artix router can be deployed either as part of an application's configuration or directly into an Artix container.

In this chapter

This chapter discusses the following topics:

Enabling Artix Routing

Overview

There are two approaches to enabling an Artix router:

- Using configuration variables.
- Using an Artix deployment descriptor.

Using configuration

You can configure an Artix router by adding the `routing` plug-in to the `orb_plugins` list, and specifying the location of the contract using the `plugins:routing:wsdl_url` entry. See "Configuring an Artix Router" on page 76 for full details.

This configuration-based approach can be used with an Artix container. Alternatively, you can also deploy a router into any Artix process. For example, this might be useful if you want to write CORBA clients and use Artix APIs.

You can also specify additional configuration variables to optimize performance. See "Optimizing Router Performance" on page 83.

Using a deployment descriptor

You can only use a deployment descriptor to define routes if you are using the container to host the router. The advantage of this approach is that you do not need a dedicated configuration scope.

Another advantage to this approach is that you can deploy additional routes into the process without stopping and restarting the host process, which would be necessary in the configuration approach.

When using the deployment descriptor approach, you must deploy each router instance separately; whereas with the configuration approach, all router instances are loaded automatically on startup. See "Deploying a Router Using a Deployment Descriptor" on page 79 for full details.

Selecting a host process

Although any Artix process can be used for Artix routing, the preferred approach is to use the Artix container as the host process.

When using the Artix container server process (`it_container`), you have the option of using either the configuration approach, or the deployment descriptor approach.

In addition, you can also use the container's client application (`it_container_admin`) to manage the deployed route.

Note: If you use an Artix client or server process to host the `routing` plug-in, you can only use configuration to specify routing details. You can not use a deployment descriptor.

Disabling a router

To undeploy a router, you must stop and restart the process hosting the router. This applies to both the configuration and deployment descriptor approach.

Using the configuration approach, you must edit the `plugins:routing:wsdl_url` entry, removing the contract describing the routes you wanted to undeploy.

Using the deployment descriptor approach, you would then either not redeploy that particular contract, or you would remove its corresponding deployment descriptor from the persistent deployment directory. See Configuring and Deploying Artix Solutions, C++ Runtime for full details.

Configuring an Artix Router

Overview

Because Artix's routing functionality is implemented as an Artix plug-in, you can make any Artix application a router by adding routing rules to its contract, and by specifying configuration settings in an Artix configuration file.

This section explains how to configure the `routing` plug-in, and specify the location of the router's contract.

Setting the orb_plugins list

Artix routers must include the `routing` plug-in name in its `orb_plugins` list, for example:

```
orb_plugins = ["xmlfile_log_stream", "soap", "at_http", ... ,
    "routing"];
```

Note: You do not need to add the `routing` plug-in if you have defined routes in a deployment descriptor (see "Deploying a Router Using a Deployment Descriptor" on page 79).

Plug-ins related to bindings, and transports are not required. These are loaded automatically when the `routing` plug-in parses the contract.

Note: The `routing` plug-in must always be the last plug-in listed in the `orb_plugins` list.

Setting the WSDL contract

You must configure the location of the contract, or contracts, that the router gets its routing information from. You can do this using the `plugins:routing:wsdl_url` variable. This variable specifies the contracts that the router parses for routing rules. The following is a simple example:

```
plugins:routing:wsdl_url="../../etc/router.wsdl";
```

The location of the contract is relative to the location from which the Artix router is started.

The following example contains multiple routing contracts:

```
plugins:routing:wsdl_url=["route1.wsdl", "../route2.wsdl",
                          "/artix/routes/route3"];
```

In this example, the router expects that `route1.wsdl` is located in the directory that it was started in, and that `route2.wsdl` is located one directory level higher.

Defining a single route in configuration

This is the simple approach used by the `routing` demos (for example, `routing\operation_based`).

Run the host process under a dedicated configuration scope. In this scope, include the `routing` plug-in name in the `orb_plugins` list, and use the `plugins:routing:wsdl_url` variable to specify the location the contract containing the routing rules.

The required configuration is illustrated in Example 21, where `demos.operation_based.router` is the scope under which the host process runs.

Example 21: *Simple Router Configuration*

```
demos {
 operation_based {
    orb_plugins = ["xmlfile_log_stream", "soap", "at_http"];

     router {
      #the routing plug-in implements the routing functionality
      orb_plugins = ["routing"];
```

Example 21: *Simple Router Configuration (Continued)*

```
      #the path to the WSDL file that includes the routing element
       plugins:routing:wsdl_url="../../etc/route.wsdl";
    };
  };
};
```

This router can then be deployed in the container server using the following command:

```
it_container -ORBname demos.operation_based.router
   -ORBdomain_name operation_based  -ORBconfig_domains_dir
   ../../etc -publish
```

Defining multiple routes in configuration

There are two approaches to using configuration to deploy multiple routes into the same host process. The first is to specify multiple routes in a single contract. Using this approach the configuration is the same as that shown in Example 21. Using this approach sacrifices the modularity of your routes for ease of configuration.

The second approach is to place your routes in multiple contracts. Using this approach you must list multiple entries for the `plugins:routing:wsdl_url` variable, as shown in the following example:

```
plugins:routing:wsdl_url= ["../../etc/route1.wsdl",
   "../../etc/route2.wsdl"];
```

In this case, each contract may include one, or more, routes. When listing multiple contracts, use the list format for specifying configuration variables

Further information

For details of optional router configuration settings, see "Optimizing Router Performance" on page 83.

For details of all the configuration options available for the `routing` plug-in, see the Artix Configuration Reference.

Deploying a Router Using a Deployment Descriptor

Overview

This section explains how to deploy a router into an Artix container using a deployment descriptor. This approach is illustrated in the `advanced\container\deploy_routes` demo.

Defining multiple routes

In the `deploy_routes` demo, the Artix container process starts under the global configuration scope defined in the `artix.cfg` configuration file.

Note: In this case, the `routing` plug-in is not loaded during startup because it is not listed in the `orb_plugins` configuration entry.

The extract shown in Example 22 is from one of the contracts used in the `advanced\container\deploy_routes` demo.

Example 22: *Deploy Routes Contract*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BaseService"
    targetNamespace="http://www.iona.com/bus/demos/router"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.iona.com/bus/demos/router"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:corba="http://schemas.iona.com/bindings/corba"
    xmlns:routing="http://schemas.iona.com/routing">

    <portType name="GoodbyeServicePortType">
       <operation name="say_goodbye">
           <input message=… name=…/>
           <output message=… name=…/>
       </operation>
    </portType>
```

Example 22: *Deploy Routes Contract (Continued)*

```
    <binding name="SOAPGoodbyeServiceBinding" type="tns:GoodbyeServicePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="say_goodbye">
            <soap:operation …/>
             …
        </operation>
    </binding>

    <binding name="CORBAGoodbyeServiceBinding" type="tns:GoodbyeServicePortType">
        <corba:binding repositoryID="IDL:GoodbyeServicePortType:1.0"/>
            <operation name="say_goodbye">
            …
           </operation>
    </binding>

    <service name="SOAPHTTPService">
        <port binding="tns:SOAPGoodbyeServiceBinding" name="SOAPHTTPPort">
            <soap:address location=…/>
        </port>
    </service>

    <service name="CORBASoapService">
        <port binding="tns:CORBAGoodbyeServiceBinding" name="CORBASoapPort">
            <corba:policy poaname=…/>
            <corba:address location=…/>
        </port>
    </service>

    <routing:route name="CorbaToSoap">
        <routing:source port="CORBASoapPort" service="tns:CORBASoapService"/>
        <routing:destination port="SOAPHTTPPort" service="tns:SOAPHTTPService"/>
    </routing:route>
</definitions>
```

The corresponding deployment descriptor is shown in Example 23.

Example 23: *Deploy Routes Deployment Descriptor*

```
<?xml version="1.0" encoding="utf-8"?>
<m1:deploymentDescriptor xmlns:m1="http://schemas.iona.com/deploy">

   <service xmlns:servicens="http://www.iona.com/bus/demos/router"> servicens:CORBASoapService
   </service>

   <wsdl_location>
    ../../routes/soap_route.wsdl
   </wsdl_location>

   <plugin>
      <name>routing</name>
      <type>Cxx</type>
      <implementation>it_routing</implementation>
      <provider_namespace>
          http://schemas.iona.com/routing
      </provider_namespace>
   </plugin>
</m1:deploymentDescriptor>
```

In the example deployment descriptor, the opening `service` element specifies the `targetNamespace` as an attribute and the source service name as the element value. This information links the deployment descriptor to a specific service. The `wsdl_location` element provides the path to the contract that includes the related route. The `plugin` element includes the information needed to load the `routing` plug-in.

In the `advanced\container\deploy_plugin` demo, each contract includes only one route. However, a contract can include multiple routes and be referenced in the `wsdl_location` element in multiple deployment descriptors. In this scenario, each deployment descriptor uniquely identifies a source service using the content in the opening `service` element.

Deploying multiple routes

In the `deploy_routes` demo, the container client application (`it_container_admin`) is used to deploy two routes, each of which is specified in a dedicated deployment descriptor file. For example:

```
it_container_admin -deploy -file
   ../../routes/deployCORBASoapService.xml
it_container_admin -deploy -file
   ../../routes/deployCORBAHTTPService.xml
```

Each deployment descriptor describes a single router, which is identified by the `targetNamespace` assigned to the contract that contains the route and the name of the source service.

Specifying persistent deployment

With the deployment descriptor approach, you can specify a persistent deployment directory. When you initially deploy each contract, a copy of the deployment descriptor is placed into this directory.

When you restart the container, it automatically redeploys all the contracts identified in these deployment descriptors. In this case, the effect is the same as the configuration approach (that is, all routes are deployed during the startup).

Further information

For more details on the Artix container, deployment descriptors, and persistent deployment, see *Configuring and Deploying, C++ Runtime*.

For working examples of the `routing` plug-in deployed in an Artix container, see any of the demos in the following directory:

`InstallDir``\samples\routing`

Alternatively, for a more advanced example, see:

`InstallDir``\samples\advanced\container\deploy_routes`

Optimizing Router Performance

Overview

This section describes how to configure the following router optimizations in an Artix configuration file:

- Setting router pass-through
- Setting CORBA bypass

Setting router pass-through

By default a router instance to passes along messages without processing if the source and destination of the route use the same binding. You can change this behavior by setting `plugins:routing:use_pass_through` to `false`.

When the router passes a message in its default pass-through mode it copies the message buffer directly from the source endpoint to the destination endpoint. This has a number of implications:

- Reference proxification does not occur.
- Request level handlers are not called.
- Server-side message level handlers are not called.
- Authentication and authorization are skipped regardless of the security settings.

If you want all messages to go through the router and be fully processed, set this variable to `false`.

Setting CORBA bypass

For CORBA integrations, you can use location forwarding to connect CORBA clients directly to CORBA servers, and thus bypass the Artix `routing` plug-in entirely.

Set the `plugins:routing:use_bypass` configuration variable to `true` to specify that the router sends CORBA `LocateReply` messages back to the client. The default is `false`.

Further information

For more information on Artix router optimizations, see the Artix Configuration Reference.

CHAPTER 11

Routing Messages Containing References

When routing messages containing endpoint references, Artix creates client proxies for the referenced endpoint. This chapter explains how to optimize router performance when routing messages containing endpoint references.

In this chapter

This chapter discusses the following topics:

Endpoint References and the Router

Overview

This section explains how the Artix router treats endpoint references when routing to client systems. For example, you can use the router to expose a service with a legacy payload and transport (CORBA/IIOP) to clients with a newer payload and transport (SOAP/HTTP).

References, client proxies, and transient servants

When endpoint references are passed across the router, a *client proxy* representation of the reference is created for the client to invoke on. The router forwards the client invocation to the server backend along with the client proxy representation. The process of creating the client proxy from the endpoint reference is called *proxification*. This process enables the router to translate between different transports and protocols. A reference of a certain type (such as CORBA) that passes through the router is automatically converted to a reference of another type (such as SOAP).

For example, take the use case where a SOAP client invokes on a SOAP/HTTP-to-CORBA router, which forwards it on to a CORBA backend. In this scenario, a client call to `MyBank::get_account()` returns an `Account` reference. The client proxy created for this reference represents a route to the backend, and this is the key element in bridging the invocation. The part of the router that invokes on this client proxy is essentially a service inside the router and is represented by a servant.

The nature of the `get_account()` invocation means that many similar `Account` references, client proxies, and servants are created in the router, thereby causing unlimited memory bloat, depending on the number of `Account` references passing through the router. The servant objects created in the router are also called *transient servants*.

Default servant model

An alternative to using transient servants is a model called the *default servant*, which maintains a template-based representation of the service and automatically redirects to the correct client proxy.

In previous versions of Artix, the router followed the transient servant model for `get_account()` style invocations. The router now uses the default servant model, which makes it more efficient and more scalable. This also means that you can manage memory issues in the router simply by setting the appropriate router configuration variables. There are no changes required to application code or WSDL contracts. For details, see "Preventing Memory Bloat in the Router" on page 88.

Note: Router proxification is available for the following bindings and transports: CORBA, SOAP, HTTP, and IIOP Tunnel.

Further information

For information on developing applications using the default servant model and transient servant model, see Developing Artix Applications in C++ and Developing Artix Applications with JAX-RPC.

Preventing Memory Bloat in the Router

Overview

Because the router creates a new client proxy for each endpoint reference that passes through it, the router can suffer from memory bloating. To prevent this bloating, you can specify the following in the router's runtime configuration:

- maximum number of proxified references in the router
- maximum number of unproxified references in the router

Maximum proxified references

You can specify the maximum number of proxified endpoint references in the router using the `plugins:routing:proxy_cache_size` configuration variable. This is the number of endpoint references that have already been converted into a client proxy and are ready for invocation.

`plugins:routing:proxy_cache_size` works in conjunction with `plugins:routing:reference_cache_size`. Having a smaller setting for `proxy_cache_size` enables the router to conserve memory, while still being ready for invocations. This is because proxified references use more resources than unproxified references. The default setting is:

```
plugins:routing:proxy_cache_size=50;
```

The router caches references on a least recently used basis in the order: proxified, unproxified. A proxified reference is demoted to an unproxified reference when the `proxy_cache_size` limit is reached. Unproxified references are promoted to proxies upon invocation.

Maximum unproxified references

You can specify the maximum number of unproxified endpoint references in the router using the `plugins:routing:reference_cache_size` configuration variable. This refers to the number of references that must be proxified before they can be invoked on.

`plugins:routing:reference_cache_size` works in conjunction with `plugins:routing:proxy_cache_size`. Having a larger setting for `reference_cache_size` enables the router to conserve memory, while still being ready for invocations, because unproxified references use less resources than proxies. The default setting is:

```
plugins:routing:reference_cache_size="1000";
```

Example banking system

For example, take a SOAP over HTTP client and CORBA server banking system, with the router deployed between the client and the server. There are 1,500 accounts in this banking system.

By default, the 50 most recently used accounts are present in the router as proxified references. The next 1000 most recently used are present as unproxified references. While the remaining 450 do not exist in the router, but can be created on-demand.

Further information

For more information on these router configuration variables, see the Artix Configuration Reference, C++ Runtime.

For more information about Artix configuration in general, see Configuring and Deploying Artix Solutions, C++ Runtime.

CHAPTER 12

Error Handling

The routing service reports errors back to the message originator.

Initialization errors

Errors that can be detected when the routing service is initializing, such as routing between incompatible endpoints and some kinds of route ambiguity, are logged and an exception is raised. This exception aborts the initialization and shuts down the service.

Runtime errors

Errors that are detected at runtime are reported as exceptions and returned to the message originator; for example “no route” or “ambiguous routes”.

The destination endpoint does not receive any notification that a message failed to be forwarded to it. If your endpoints require such notification, you need to implement a mechanism to deliver the notification outside the scope of the routed operation.

Index

S

T

U

X