



IVI Technologies

Integrity Value Innovation

Stylus XQuery[®]

User's Guide and Reference

Release 5.0
August 2024

@Copyright 2024 IVI Technologies

These materials and all IVI Technologies products are copyrighted and all rights are reserved by IVI Technologies Corporation. The information in these materials is subject to change without notice and IVI Technologies assumes no responsibilities for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Stylus Studio, Stylus XML Converters, Stylus XQuery, XML Pipeline Server are trademarks or service marks of IVI Technologies Corporation and/or its subsidiaries or affiliates in the US and other countries Java is a trademark of Oracle and its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Third Party Acknowledgments: There are no products in the Stylus XML Converters release that include third party component covered by licenses that require documentation notices be provided.

1999-2006 the Apache Software Foundation. All rights reserved.

XercesJ developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

FOP developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

Axis developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

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

Files that are subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.dstc.com>. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2002. All rights reserved.

Pathan developed by DecisionSoft Limited. Copyright © 2001 DecisionSoft Limited. All rights reserved.

Software developed by Thai Open Source Software Center Ltd. Copyright © 2001-2003, Thai Open Source Software Center Ltd. All rights reserved.

IBM ICU developed by IBM. Copyright © 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.

Software developed by Kevin Atkinson. Copyright © 2000-2004, by Kevin Atkinson. All rights reserved.

Aspell 0.60.2, from the Free Software Foundation, Inc. (<http://www.fsf.org/>), which is subject to the GNU Lesser General Public License Version 2.1 (<http://www.gnu.org/licenses/lgpl.html>). Software distributed under this license is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license.

Software developed by xqDoc.org. Copyright © 2005 Elsevier, Inc. All rights reserved.

Software developed by Info-ZIP. Copyright © 1990-2004 Info-ZIP. All rights reserved. For the purposes of this copyright and license, "Info-ZIP" is defined as the following set of individuals: Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Christian Spieler, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White. Info-ZIP software is provided "as is", without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

Software developed by Tim Bray and Sun Microsystems and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Copyright © 2004 Tim Bray and Sun Microsystems. All rights reserved.

Software developed by Saxonica Limited and is distributed on an "AS IS" basis WITHOUT WARRANTY OF ANY KIND, either express or implied. Copyright © 2005 Saxonica Limited. All rights reserved.

Software developed by The Anti-Grain Geometry Project. Copyright © 2002-2005 Maxim Shemanarev (McSeem). This software is provided "as is" without express or implied warranty, and with no claim as to its suitability for any purpose.

Stylus XML Converters. Copyright 2024 IVI Technologies Corporation and/or its subsidiaries or affiliates. All rights reserved.

Stylus XQuery. Copyright 2024 IVI Technologies Corporation and/or its subsidiaries or affiliates. All rights reserved.

Stylus XML Converters includes:

Software developed by World Wide Web Consortium. Copyright (c) 1998-2003 World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

Software developed by World Wide Web Consortium. Copyright (c) 1998-2000 World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

Software developed by JSON.org. Copyright (c) 2002 JSON.org. All rights reserved.

Stylus XQuery includes:

XQJ 225 XQuery API for Java 1.0 Reference Implementation. Copyright (c) 2003 -2007 Oracle. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED, IMPLIED OR STATUTORY WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL ORACLE OR ITS LICENSORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ORACLE IS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

August 2024

Table of Contents

Contents

Stylus XQuery [®]	1
Table of Contents	5
List of Tables	19
Preface.....	25
What Is Stylus XQuery [®] ?	25
Using This Book.....	26
About the Product Documentation.....	29
Typographical Conventions.....	31
Contacting Technical Support	33
1 Quick Start.....	35
Getting Started with Stylus XQuery	35
Using the Command Line Utility.....	40
Additional Resources	45
2 Introduction.....	47
What Is Stylus XQuery [®] ?	47
What Is XQuery?.....	48
What Is XQJ?	53
Stylus XQuery [®] Architecture	55
Using Stylus XML Converters TM	56
Using Stylus Studio [®]	57
3 Tutorial: Using Stylus XQuery [®]	59
Specifying the XQuery Version	60
Configuring Connections	61
Executing Queries	62
Querying Data from XML Files or DOM Trees	65
Joining Data from XML and Relational Sources.....	67
Returning Results with Java XML APIs	69
DOM	70
SAX	71
Preparing XQuery Statements.....	72
Updating Data in Relational Databases.....	74
4 Tutorial: Using XQuery.....	75

Finding XML Nodes: Path Expressions	76
Creating XML: XML Constructors	83
Restructuring Data: FLWOR Expressions	84
Grouping Data	92
Summary	100
5 Tutorial: The XQuery Update Facility	101
Support Overview	102
XUF Examples	103
Storing Query Results	104
Replacing Node Values	105
Inserting a New Node	106
Renaming a Node	108
Transforming Query Results	109
Updating Data Sources	112
6 Understanding Data Sources and Connections	115
Using Data Sources in Queries	115
Choosing a Connection Method	123
Configuring Connections Explicitly	123
Configuring Connections Using JNDI	127
DDXQDataSource and DDXQJDBCConnection Properties	128
DDXQDataSource Properties	128
Specifying Connection URIs	141
7 Securing Data Source Connections	145
About Authentication	145
Using Kerberos Authentication	147
Using NTLM Authentication	159
Data Encryption Across the Network	163
8 Improving Performance	177
Querying Large XML Documents	177
Using Comparisons	189
Understanding Compensation	191
Using Query Pooling	192
Using Connection Pooling	193
9 Building a Web Service	201
XQueryWebService Framework Overview	201
XQueryWebService Framework Architecture	204
Example – Employee Lookup	211
Specifying a Database Connection	213
Choosing an Interface for Web Service Access	216
Tools for Testing Web Service Operations	219
Generating WSDL	221
Using WSDL Service References	223
10 Building a Web Service Client	229

Overview.....	229
Stylus HTTP Functions	231
Example: Web Service Client Comparison.....	244
HTTP Function Request and Response XML Schemas	247
11 Support for Relational Databases	251
Querying Relational Data	251
Querying XML Type Data.....	263
Updating Relational Data	267
Understanding the Transactional Behavior of Stylus XQuery Updates	270
12 Using Advanced Features.....	275
Using Option Declarations and Extension Expressions	275
Querying Multiple Files in a Directory	288
Querying ZIP, JAR, and MS Office Files	292
Using URI Resolvers	295
Analyzing EDI to XML Conversions.....	299
Generating XQuery Execution Plans	307
Specifying Collations.....	312
Using External Functions	315
A XQuery Support.....	339
2 Basics	341
3 Expressions	347
4 Modules and Prologs	357
5 Conformance.....	358
Namespaces	360
B Functions and Operators.....	361
C Built-in Functions and Options.....	389
Stylus XQuery Built-In Functions	389
HTTP Functions <request> Element	433
Stylus XQuery Options.....	437
Namespaces	437
D Serialization Support.....	439
Overview.....	439
Serialization Methods	439
E Database Support	445
Supported Databases	445
Data Type Mappings.....	447
DB2	448
Supported XQuery Atomic Types.....	459
Database Connection Properties	460
DB2	460
Database-Specific Query Functions	483
F XUF Support.....	501
2 Extensions to XQuery 1.0	502

5 Conformance	506
G XQJ Support	507
Java Package Name	508
XQConnection Interface	508
XQDataFactory	510
XQDataSource Interface	512
XQDynamicContext Interface	514
XQExpression Interface	516
XQItem Interface	517
XQItemAccessor Interface	517
XQItemType Interface	519
XQMetaData Interface	521
XQPreparedExpression Interface	522
XQResultItem Interface	523
XQResultSequence Interface	524
XQSequence Interface	524
XQSequenceType Interface	526
XQStaticContext Interface	526
Exception Handling	529
Multi-Threading Support	530
Accessing XML Results	530
DOM	531
SAX	531
Support of Deferred Binding	532
XQuery Types Supported by XQJ get Methods	533
Retrieving and Binding XQuery Data Model Instances	534
H Examples	539
Required Software	539
Configuring Your Environment to Run the Examples	540
About the Examples	541
I Troubleshooting	553
Logging XQJ Calls with Stylus Spy TM	553
Java Logging	562
Resolving fn:collection Errors	564
Resolving Static Type Errors	573
Index	581
A	581
C	582
D	583
E	586
F	587
G	588

H.....	588
I.....	588
J.....	589
K.....	590
L.....	590
M.....	590
N.....	591
O.....	591
P.....	592
Q.....	593
R.....	594
S.....	594
T.....	597
U.....	598
V.....	598
W.....	598
X.....	599
Z.....	602

List of Tables

Table 1-1.	Command Line Utility Options	41
Table 6-1.	DDXQDataSource Properties	128
Table 6-2.	DDXQJDBCConnection Properties	136
Table 7-1.	Authentication Methods Supported by Stylus XQuery	146
Table 7-2.	Kerberos Authentication Requirements	148
Table 7-3.	NTLM Authentication Requirements	159
Table 7-4.	Data Encryption Methods Supported by Stylus XQuery	164
Table 7-5.	EncryptionMethod Property and Microsoft SQL Server Configurations 173	
Table 10-1.	Recognized Mime Types and Associated Encodings	241
Table 10-2.	<request> Element cookie-policy Parameters	243
Table 11-1.	Isolation Level Support	273
Table 12-1.	Global Option Declarations	276
Table 12-2.	Relational Option Declarations	279
Table 12-3.	Database-Specific Option Declarations	283
Table 12-4.	Extension Expressions	287
Table 12-5.	Mapping Types Between Java and XQuery	320
Table A-1.	XQuery Expression Context	342
Table A-2.	XQuery Processing Model	345
Table A-3.	Error Handling	345
Table A-4.	XQuery Documents	346
Table A-5.	XQuery Types	346
Table A-6.	Comments	347

Table A-7.	XQuery Primary Expressions	348
Table A-8.	XQuery Path Expressions.	349
Table A-9.	XQuery Sequence Expressions.	350
Table A-10.	XQuery Arithmetic Expressions.	350
Table A-11.	XQuery Comparison Expressions.	351
Table A-12.	XQuery Logical Expressions.	351
Table A-13.	XQuery Constructors	352
Table A-14.	XQuery FLWOR Expressions	353
Table A-15.	XQuery Ordered and Unordered Expressions	353
Table A-16.	XQuery Conditional Expressions.	354
Table A-17.	XQuery Quantified Expressions	354
Table A-18.	XQuery Expressions on Sequence Types.	355
Table A-19.	XQuery Validate Expressions	356
Table A-20.	XQuery Extension Expressions	356
Table A-21.	XQuery Modules and Prologs	357
Table A-22.	XQuery Optional Features	358
Table A-23.	Predefined Namespaces	360
Table B-1.	XQuery Accessor Functions	362
Table B-2.	XQuery Error Function.	363
Table B-3.	XQuery Trace Function	363
Table B-4.	XQuery Constructor Functions for XML Schema Built-In Types	364
Table B-5.	A Special Constructor Function for xs:dateTime	366
Table B-6.	XQuery Operators on Numeric Values	367
Table B-7.	XQuery Comparison Operators on Numeric Values	368
Table B-8.	XQuery Functions on Numeric Values	368
Table B-9.	XQuery Functions to Assemble and Disassemble Strings	369

Table B-10. XQuery Functions for Equality and Comparison of Strings	369
Table B-11. XQuery Functions on String Values.....	370
Table B-12. XQuery Functions Based on Substring Matching	371
Table B-13. String Functions That Use Pattern Matching	372
Table B-14. XQuery anyURI Functions	372
Table B-15. XQuery Boolean Constructor Functions	373
Table B-16. XQuery Operators on Boolean Values.....	373
Table B-17. XQuery Functions on Boolean Values	374
Table B-18. Functions on Duration, Date, and Time Data Types.....	374
Table B-19. XQuery Comparisons of Duration, Date, and Time Values	375
Table B-20. XQuery Component Extraction Functions	377
Table B-21. XQuery Arithmetic Operators on Durations	378
Table B-22. Functions for Timezone Adjustment on dateTime, date, and time Values	379
Table B-23. Operators for Adding and Subtracting Durations from dateTime, date, and time	379
Table B-24. Constructor Functions for QNames	381
Table B-25. XQuery Operators and Functions Related to QNames	381
Table B-26. XQuery Comparisons of base64Binary and hexBinary Values	382
Table B-27. XQuery Operators on NOTATION	383
Table B-28. XQuery Functions and Operators on Nodes.....	383
Table B-29. XQuery General Functions and Operators on Sequences	384
Table B-30. XQuery Functions that Test Cardinality on Sequences	385
Table B-31. XQuery Functions and Operators on Equals, Union, Intersection, and Except	385
Table B-32. XQuery Aggregate Functions.....	386
Table B-33. XQuery Functions and Operators That Generate Sequences	386
Table B-34. XQuery Context Functions.....	387

Table 12-6.	Common Picture String Specifiers	399
Table 12-7.	ddtek:format-number Function Parameters	403
Table 12-8.	Function Request Parameters	433
Table C-1.	Predefined Namespaces	437
Table D-1.	Serialization Parameters	440
Table D-2.	Formats Supported by the Stylus XML Converters	442
Table 12-9.	Stylus XQuery Relational Database Support	445
Table E-1.	DB2 Data Types	448
Table E-2.	Informix Data Types	449
Table E-3.	MySQL Enterprise Data Types	451
Table E-4.	Oracle Data Types	452
Table E-5.	PostgreSQL Data Types	454
Table E-6.	Microsoft SQL Server Data Types	455
Table E-7.	Sybase Data Types	457
Table E-8.	Predefined XQuery Atomic Types	459
Table E-9.	DB2 Connection Properties	461
Table E-10.	Informix Connection Properties	466
Table E-11.	Microsoft SQL Server Connection Properties	468
Table E-12.	MySQL Enterprise Connection Properties	470
Table E-13.	Oracle Connection Properties	472
Table E-14.	Oracle Connection Property Mappings to tnsnames.ora Connect Descriptor Parameters	478
Table E-15.	Sybase Connection Properties	481
Table 12-10.	Ways to Query XML Data on DB2	484
Table F-1.	Extensions to the Processing Model	502
Table F-2.	Extensions to the Prolog	503
Table F-3.	Extensions to the Static Context	503

Table F-4.	New Kinds of Expressions	504
Table F-5.	Extensions to Built-in Function Library	505
Table F-6.	Extensions to Built-in Function Library	505
Table G-1.	XQConnection Method Summary.....	508
Table G-2.	XQDataFactory Method Summary.....	510
Table G-3.	XQDataSource Method Summary.....	513
Table G-4.	XQDynamicContext Method Summary.....	514
Table G-5.	XQExpression Method Summary	516
Table G-6.	XQItem Method Summary.....	517
Table G-7.	XQItemAccessor Method Summary.....	517
Table G-8.	XQItemType Method Summary.....	519
Table G-9.	XQMetaData Method Summary.....	521
Table G-10.	XQPreparedExpression Method Summary	522
Table G-11.	XQResultItem Method Summary	523
Table G-12.	XQResultSequence Method Summary.....	524
Table G-13.	XQSequence Method Summary	524
Table G-14.	XQSequenceType Method Summary	526
Table G-15.	XQStaticContext Method Summary	527
Table G-16.	XQuery Types Supported for XQJ get Methods	533
Table G-17.	XQJ bind Methods and Resulting XQuery Data Model Instances.....	534
Table G-18.	Mapping XQuery Data Model Instances to Java Objects	536
Table I-1.	Stylus Spy Attributes	557

Preface

This book is your guide and reference to Stylus XQuery® from IVI Technologies and describes how to use Stylus XQuery to access and update both XML and relational sources, and to return XML results. This book provides information about the following topics:

- Using Stylus XQuery to query both XML and relational sources, and return XML results
- Using Stylus XQuery to update relational sources
- Stylus XQuery support for XQuery and the XQuery API for Java™ (XQJ)
- Examples and tutorials that show how you can use Stylus XQuery in your environment
- Using Stylus Spy™ for XQJ, a tracing and logging utility for troubleshooting

What Is Stylus XQuery®?

Stylus XQuery is an XQuery processor that enables developers to access and query XML, relational data, SOAP messages, EDI, or a combination of data sources, and, in addition, provides full update support for relational data. Stylus XQuery supports the XQuery API for Java (XQJ) API, and is easily embeddable into any Java program; it does not require any other product or application server, and has no server of its own. It is recommended for developers who need to combine and efficiently process XML, relational, and legacy data formats in application scenarios such as data integration,

XML-based data exchange, XML-driven web sites, and XML publishing. Stylus XQuery vastly simplifies and enhances the performance of combining and processing different types of data (relational, XML, EDI, and more) in heterogeneous environments and, thus, enables developers to build and deploy high-performance applications quickly and efficiently.

Using This Book

This book assumes that you are familiar with your operating system and its commands; the concept of directories; the management of user accounts and security access; and your network configuration.

This book contains the following chapters:

- [Chapter 1 “Quick Start” on page 35](#) provides basic information for getting started with Stylus XQuery immediately after installation.
- [Chapter 2 “Introduction” on page 47](#) introduces Stylus XQuery, XQuery, XQJ, and development tools. In addition, it provides examples of XQuery and a Java application that uses XQJ to execute a query.
- [Chapter 3 “Tutorial: Using Stylus XQuery®” on page 59](#) shows how to use Stylus XQuery and XQJ in your Java application to perform tasks that allow you to process queries that access XML and relational data sources, and return XML results.
- [Chapter 4 “Tutorial: Using XQuery” on page 75](#) focuses on the three major capabilities of XQuery that make it distinctive, and which are fundamental to processing and creating XML: path expressions, XML constructors, and FLWOR expressions.

- [Chapter 5 “Tutorial: The XQuery Update Facility” on page 101](#) describes the XQuery Update Facility (XUF), an extension of the XQuery language that allows making changes to data that are manipulated inside the XQuery.
- [Chapter 6 “Understanding Data Sources and Connections” on page 115](#) provides conceptual information about Stylus XQuery data sources and connections, and instructions for configuring them.
- [Chapter 7 “Securing Data Source Connections” on page 145](#) describes how to implement supported authentication and data encryption securing methods.
- [Chapter 8 “Improving Performance” on page 177](#) describes information about performance that you should consider when working with Stylus XQuery.
- [Chapter 9 “Building a Web Service” on page 201](#) describes how to expose your XQuery as a Web service.
- [Chapter 10 “Building a Web Service Client” on page 229](#) describes how to use Stylus XQuery built-in HTTP functions to build a Web service client.
- [Chapter 11 “Support for Relational Databases” on page 251](#) explains Stylus XQuery’s support of relational databases. Specifically, it describes support of XML-typed data and how Stylus XQuery generates SQL.
- [Chapter 12 “Using Advanced Features” on page 275](#) provides information about the following advanced features: option declarations, extension expressions, URI resolvers, and collations.
- [Appendix A “XQuery Support” on page 339](#) describes how Stylus XQuery supports XQuery 1.0 and 1.1 expressions.
- [Appendix B “Functions and Operators” on page 361](#) describes how Stylus XQuery supports XQuery functions and operators.

- [Appendix C “Built-in Functions and Options” on page 389](#) describes built-in Stylus XQuery functions and options and how to use them to process XQuery results.
- [Appendix E “Database Support” on page 445](#) describes the relational databases that Stylus XQuery supports, including XML type mappings for relational data and supported connection properties.
- [Appendix F, “XUF Support” on page 501](#) describes how Stylus XQuery supports the XQuery Update Facility (XUF) expressions, functions, optional feature. It also describes built-in Stylus XQuery functions that support XUF.
- [Appendix G “XQJ Support” on page 507](#) describes how Stylus XQuery supports XQJ classes, interfaces, and methods. It also describes serialization, multi-threading, accessing XML results, and mapping data types.
- [Appendix H “Examples” on page 539](#) explains the example Java applications that are shipped with Stylus XQuery and provides instructions for setting up and running them.
- [Appendix I “Troubleshooting” on page 553](#) provides valuable Stylus XQuery troubleshooting information, including how to use Stylus Spy, a development tool for tracking XQJ calls.

NOTE: This book refers the reader to Web pages for more information about specific topics, including Web pages that are not maintained by IVI Technologies. Because it is the nature of Web content to change frequently, IVI Technologies can guarantee only that the URLs referenced in this book were correct at the time the book was produced.

About the Product Documentation

The Stylus XQuery library consists of the following books:

- *Stylus XQuery Installation Guide* describes the requirements and procedures for installing Stylus XQuery.
- *Stylus XQuery User's Guide and Reference* provides information about using Stylus XQuery to access both XML and relational sources.

Both of these books are available in HTML and PDF format. By default, the HTML version is installed during a normal installation of Stylus XQuery. The PDF version is an optional installation. If you choose to install the PDF version, the books are installed in the books/ddxquery subdirectory of the Stylus XQuery installation directory.

HTML Version

Both of the Stylus XQuery books are placed on your system as HTML-based online Help during a normal installation of the product. The Help system is located in the /help subdirectory of the product installation directory. To use the Help, you must have one of the following browsers installed:

- Internet Explorer 5.x or higher
- Netscape 4.x, 6.1, or higher
- FireFox 1.0 or higher

You can access the Help system by navigating to the /help subdirectory of the product installation directory and opening the following file from within your browser:

`install_dir/help/help.htm`

where `install_dir` is the path to your product installation directory.

After the browser opens, the left pane displays the Table of Contents, Index, and Search tabs for the entire documentation library. When you have opened the main screen of the Help system in your browser, you can bookmark it in the browser for quick access later.

NOTE: Security features set in your browser can prevent the Help system from launching. In this case, a security warning message is displayed. Often, the warning message provides instructions for unblocking the Help system for the current session. To allow the Help system to launch without encountering a security warning message, you can modify the security settings in your browser. Check with your system administrator before disabling any security features.

PDF Version

Product documentation is also provided in PDF format, which allows you to view it, perform text searches, and print it. You can view the PDF documentation using Adobe Reader. The PDF documentation is available on the product CD, as a product installation component, and also on the IVI Technologies Web site:

<https://www.xquery.com/documentation>

You can download the entire Stylus XQuery library as a compressed file. When you uncompress the file, the library appears in the correct directory structure.

If you want to copy the documentation library from the product CD, you must maintain the directory structure that is on the CD.

- **To copy all product books**, copy the entire \books directory to your local or network drive.
- **To copy a specific set of books**, copy that book set's directory structure (beneath the \books directory) to your local or network drive. For example, in the case of:

\books\product_name

you would copy the entire \product_name directory.

Maintaining the correct directory structure allows cross-book text searches and cross-references. If you download or copy the books individually outside of their normal directory structure, their cross-book search indexes and hyperlinked cross-references to other books will not work. You can view a book individually, but it will not automatically open other books to which it has cross-references.

To help you navigate the library, a file named books.pdf is provided. This file lists each online book provided for the product. We recommend that you open this file first and, from this file, open the book you want to view.

Typographical Conventions

This book uses the following typographical conventions:

Convention	Explanation
<i>italics</i>	Introduces new terms with which you may not be familiar, and is used occasionally for emphasis.
bold	Emphasizes important information. Also indicates button, menu, and icon names on which you can act. For example, click Next .

Convention	Explanation
UPPERCASE	Indicates the name of a file. For operating environments that use case-sensitive file names, the correct capitalization is used in information specific to those environments. Also indicates keys or key combinations that you can use. For example, press the ENTER key.
monospace	Indicates syntax examples, values that you specify, or results that you receive.
<i>monospaced italics</i>	Indicates names that are placeholders for values that you specify. For example, <i>filename</i> .
forward slash /	Separates menus and their associated commands. For example, Select File / Copy means that you should select Copy from the File menu. The slash also separates directory levels when specifying locations under UNIX and Linux.
vertical rule	Indicates an "OR" separator used to delineate items.
brackets []	Indicates optional items. For example, in the following statement: SELECT [DISTINCT], DISTINCT is an optional keyword. Also indicates sections of the Windows Registry.
braces { }	Indicates that you must select one item. For example, {yes no} means that you must specify either yes or no.
ellipsis . . .	Indicates that the immediately preceding item can be repeated any number of times in succession. An ellipsis following a closing bracket indicates that all information in that unit can be repeated.
1.1	Identifies a feature or functionality that is supported only for XQuery 1.1.

Contacting Technical Support

IVI Technologies offers a variety of options to meet your technical support needs. Please visit our Web site for more details and for contact information:

<https://www.xquery.com/support>

The IVI Technologies Web site provides the latest support information through our global service network. The support program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

To obtain technical support for an evaluation copy of the product, go to:

<https://www.xquery.com/download>

or contact your sales representative.

When you contact us for assistance, please provide the following information:

- The serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a support contract, the support representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full customer information, including location.
- The product and the version that you are using.
- The type and version of the operating system where you have installed your product.

- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be recreated.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

I Quick Start

This quick start provides basic information for getting started with Stylus XQuery immediately after installation. It covers the following topics:

- ["Getting Started with Stylus XQuery"](#)
- ["Using the Command Line Utility"](#)
- ["Additional Resources"](#)

Getting Started with Stylus XQuery

This section shows you how to get up and running with Stylus XQuery. It covers the following topics:

- ["1. Setting the CLASSPATH"](#)
- ["2. Configuring Connections"](#)
- ["3. Developing a Java Application that Executes a Query"](#)

I. Setting the CLASSPATH

The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate Stylus XQuery on your computer. Only one Stylus XQuery jar file, `ddxq.jar`, must be defined in your CLASSPATH. If `ddxq.jar` is not defined in your CLASSPATH, you receive a `ClassNotFoundException` exception when trying to use Stylus XQuery.

Set your CLASSPATH to include:

```
install_dir/lib/ddxq.jar
```

where *install_dir* is the path to your Stylus XQuery installation directory.

NOTE: If you are connecting to PostgreSQL, you must also add the PostgreSQL JDBC driver jar file to the CLASSPATH. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

2. Configuring Connections

Stylus XQuery provides multiple ways to configure connections to XML data sources and relational data sources (see [“Choosing a Connection Method” on page 123](#)). This section shows how to use XQJ to create a DDXQDataSource instance in your Java application explicitly.

XML Data Source Connections

If your Java application contains queries that access an XML file, you can directly access the file as shown in the following XQJ code example, where the location and name of the XML file is specified as a parameter of fn:doc(), an XQuery function.

```
DDXQDataSource ds = new DDXQDataSource();
XQConnection conn = ds.getConnection();
conn.createExpression().executeQuery("doc('path_and_filename')");
```

Relational Data Source Connections

How you configure connection information for relational databases using XQJ depends on whether you are accessing a single database or multiple databases. This section shows how to configure connection information to access a single database.

(For information about accessing multiple databases, see [“Configuring Connections Explicitly” on page 123.](#))

To configure a single relational data source connection, use the DDXQDataSource class as shown in the following XQJ code example. This example specifies a connection URI (represented by “*URL*”) for the relational data source that you want to access and the user ID and password required to access the relational data source.

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("URL");
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

The format of the connection URL depends on whether you are using a built-in JDBC driver or a third-party driver, and the database you are connecting to. See [“Specifying Connection URIs” on page 141](#) for details.

See [“Sample Connection URIs” on page 37](#) for examples of the minimum information, including any required connection properties, that you must specify in a connection URL.

Sample Connection URIs

The following URIs are examples of the minimum information that must be specified in a connection URI.

DB2 for Linux/UNIX/Windows

```
jdbc:xquery:db2://server_name:50000;databaseName=your_database
```

DB2 for z/OS and iSeries

```
jdbc:xquery:db2://server_name:446;locationName=db2_location
```

Informix

```
jdbc:xquery:informix://server_name;1526;InformixServer=dbserver_name
```

Microsoft SQL Server

```
jdbc:xquery:sqlserver://server_name:1433
```

MySQL Enterprise

```
jdbc:xquery:mysql://server_name
```

Oracle

```
jdbc:xquery:oracle://server_name:1521
```

PostgreSQL

```
jdbc:postgresql:your_database
```

Sybase

```
jdbc:xquery:sybase://server_name:5000
```

3. Developing a Java Application that Executes a Query

Using Stylus XQuery, a Java application uses XQJ to execute a query. The Java package name of the XQJ classes is:

```
javax.xml.xquery
```

The Java class name of the Stylus XQuery implementation of the XQJ standard interface, `XQDataSource`, is:

```
com.ddtek.xquery.xqj.DDXQDataSource
```

The following sample Java code illustrates the basic steps that an application would perform to execute an XQuery expression using Stylus XQuery. This example accesses a Microsoft SQL Server data source. To simplify the example, this code does not include error handling.

```

// import the XQJ classes
import javax.xml.xquery.*;
import com.ddtek.xquery.xqj.DDXQDataSource;

// establish a connection to a relational data source
// specify the URL and the user ID and password
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
XQConnection conn = ds.getConnection("myuserid", "mypswn");
// create an expression object that is used to execute a query
XQExpression xqExpression = conn.createExpression();

// the query
String es = "for $h in collection('holdings')/holdings " +
            "where $h/stockticker='AMZN' " +
            "return $h";

// execute the query
XQResultSequence result = xqExpression.executeQuery(es);
result.writeSequence(System.out, null);

// free all resources
result.close();
xqExpression.close();
conn.close();

```

NOTE: XQJ examples are shipped with the product and are located in the /examples subdirectory in the Stylus XQuery installation directory.

Using the Command Line Utility

The Stylus XQuery command line utility allows you to quickly run and test XQueries through a console window.

To invoke this utility, enter the following command at a prompt from the `/lib` subdirectory of your Stylus XQuery installation directory (for example, `ddxq/lib`):

```
java -jar ddxq.jar
```

Alternatively, you can specify the path to the `lib` directory in the command line, for example:

```
java -jar ddxq/lib/ddxq.jar
```

NOTE: If your XQuery needs to locate classes other than the Stylus XQuery classes – if you are specifying a custom URI resolver, for example – you must perform one of the following actions:

- Set your `CLASSPATH` to include the path to the jar files or directories for these classes and invoke the utility using the following command:

```
java com.ddtek.xquery.Query
```

NOTE: If you are connecting to PostgreSQL, you must add the PostgreSQL JDBC driver jar file to the `CLASSPATH` in addition to `ddxq.jar`. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

- Add the class path to the command line:

```
java -cp c:\myClasses com.ddtek.xquery.Query
```

See [Example 8](#).

NOTE: If you are connecting to PostgreSQL, you must add the PostgreSQL JDBC driver jar file to the `CLASSPATH` in addition

to ddxq.jar. Refer to your PostgreSQL JDBC driver documentation for the name of the jar file.

The following table lists the options available for the utility.

Table 1-1. Command Line Utility Options

Option	Description
<code>-cr classname</code>	Specifies the CollectionURIResolver class to use. See “Collection URI Resolvers” on page 298 . See the NOTE on page 40 about setting your CLASSPATH for custom URI resolvers.
<code>-e [xhtml xml]</code>	Generates an XQuery execution plan and, optionally, specifies the format of the plan. If a format is not specified, XHTML is generated. See “Generating XQuery Execution Plans” on page 307 for an explanation of execution plans.
<code>-jdbc jdbcurl</code>	Specifies a connection URI. See “Relational Data Source Connections” on page 36 . NOTE: On UNIX and Linux, the value for this option must be enclosed with double quotes, for example: <pre>java -jar ddxq.jar -jdbc "jdbc:xquery:sqlserver://localhost:1433 ;databaseName=pubs;user=sa"</pre>
<code>-mr classname</code>	Specifies the ModuleURIResolver class to use. See “Library Module URI Resolvers” on page 296 . See the NOTE on page 40 about setting your CLASSPATH for custom URI resolvers.
<code>-noext</code>	Disallows calls to Java methods.
<code>-o filename</code>	Sends results (output) to specified file.

Table I-1. Command Line Utility Options

Option	Description
<code>-option</code> <code>property=value</code>	Specifies XQuery or JDBC global options. See “Using Option Declarations and Extension Expressions” on page 275 for more information.
<code>-prop</code> <code>property=value</code>	Specifies data source and connection options. See “DDXQDataSource and DDXQJDBCConnection Properties” on page 128 for more information.
<code>-p</code>	Displays a stack trace in case of an exception.
<code>-r classname</code>	Specifies the URIResolver class to use. See “Document URI Resolvers” on page 295. See the NOTE on page 40 about setting your CLASSPATH for custom URI resolvers.
<code>-s file URI</code>	Specifies an initial context item in the form of a file name or a URI.
<code>-t</code>	Displays version and timing information.
<code>-u</code>	Enables automatic updating of sources. See “Updating Data Sources” on page 112 for more information.
<code>-version</code>	Display version information.
<code>-?</code>	Displays the help for the command-line utility.
<code>param=value</code>	Specifies a query string parameter and its value.
<code>#param=value</code>	Specifies a query number parameter and its value. On UNIX and Linux, the value for this option MUST be enclosed with double quotes, for example: <code>java -jar ddxq.jar q.xq "#i=2"</code>

Table 1-1. Command Line Utility Options

Option	Description
<code>+param=value</code>	Specifies a query document parameter and its value.
<code>!option=value</code>	Specifies a serialization option and its value. See “Serialization Support” on page 439 for a list of serialization options.

Example 1: Executes a Simple XQuery

This example executes the simple query {2+5}.

```
java -jar ddxq.jar {2+5}
```

Example 2: Retrieves Values from an Initial Context Item

This example retrieves all values for UserId from the initial context item users.xml.

```
java -jar ddxq.jar -s ../../examples/xml/users.xml
{ //users/UserId }
```

Example 3: Retrieves Values and Writes Them to a File

This example retrieves all values for UserId and writes the results to a file named out.xml.

```
java -jar ddxq.jar -o out.xml
{ doc(' ../../examples/xml/users.xml' ) /users/UserId }
```

Example 4: Executes an XQuery in a File

This example executes the XQuery contained in the file myXQuery.xq using the initial context item input.xml.

```
java -jar ddxq.jar -s input.xml myXQuery.xq
```

Example 5: Binds a Query Document Parameter

This example executes the XQuery contained in the file `myXQuery.xq` binding the query document parameter `inputDoc` to the `input.xml` document.

```
java -jar ddxq.jar myXQuery.xq +inputDoc=input.xml
```

Example 6: Binds a Query String Parameter and Sets an Option

This example executes the XQuery contained in the file `myXQuery.xq` binding the query string parameter `param1` to the character string `Jonathan` and setting the serialization option `indent` to `yes` so that results are indented.

```
java -jar ddxq.jar myXQuery.xq param1=Jonathan !indent=yes
```

Example 7: Accesses a Relational Data Source

This example executes the XQuery contained in the file `myXQuery2.xq` that accesses a relational data source. See the [NOTE](#) about specifying connection URLs.

```
java -jar ddxq.jar -jdbc  
"jdbc:xquery:sqlserver://localhost:1433;databaseName=pubs;  
user=sa" myXQuery2.xq
```

Example 8: Specifies a Document URI

This example retrieves all values for `UserId`, specifies a document URI, and writes the results to a file named `out.xml`.

```
java -cp c:\myClasses com.ddtek.xquery.Query  
-r myURIResolver -o out.xml {doc('users.xml')/users/UserId}
```

Additional Resources

In addition to this quick start, you might find these resources useful:

- For complete information about the many Stylus XQuery features, read the other chapterstotics in this bookhelp.
- For information about product requirements, refer to "[System and Product Requirements](#)" in the *Stylus XQuery Installation Guide*.
- For information about getting started with the examples shipped with Stylus XQuery, see [Appendix H "Examples" on page 539](#).
- For information about using the Stylus XML Converters and Stylus Studio, refer to:
<https://www.xmlconverters.com>
<https://www.stylusstudio.com>

2 Introduction

This chapter introduces Stylus XQuery, XQuery, XQJ, and associated development tools. In addition, it provides examples of queries and a Java application that uses XQJ to execute a query.

What Is Stylus XQuery®?

Stylus XQuery® is an XQuery processor that enables developers to access and query XML, relational data, SOAP messages, EDI, or a combination of data sources, and, in addition, provides full update support for relational data. Stylus XQuery supports the XQuery for Java™ (XQJ) API, and is easily embeddable into any Java program; it does not require any other product or application server, and has no server of its own. It is recommended for developers who need to combine and efficiently process XML, relational, and legacy data formats in application scenarios such as data integration, XML-based data exchange, XML-driven web sites, and XML publishing. Stylus XQuery vastly simplifies and enhances the performance of combining and processing different types of data (relational, XML, legacy, EDI, and more) in heterogeneous environments and, thus, enables developers to build and deploy high-performance applications quickly and efficiently.

See [Chapter 3 “Tutorial: Using Stylus XQuery®” on page 59](#) for a tutorial that shows how to use Stylus XQuery and XQJ in your Java application. This tutorial explains tasks that allow you to process queries that access XML and relational data sources, and return XML results.

What Is XQuery?

XQuery is a query language for XML. In the same way that SQL is used to query relational tables, XQuery is used to query XML or anything for which a logical XML view can be defined. Typically, SQL queries create tables to represent the result of a query, and XQuery queries create XML to represent the result of a query. The resulting XML can be as complex as necessary. For example, the result of a query may be a complex document such as an inventory report, a document with dynamic content, or a SOAP message. The result of an XQuery can also be as simple as a single integer; for example, a query might count the number of items that satisfy a condition. In this book, we use the term *XML results* to refer to the results of any XQuery query.¹

XQuery goes beyond the functionality of relational query languages, and includes support for many features not found in the SQL language. Just as SQL is a relational query language and Java is an object-oriented language, XQuery is often thought of as a native XML programming language. In XQuery, the only complex data structure is XML, and the operations that are regularly needed for processing XML are directly supported in a convenient manner.

XQuery can easily search any XML structure with path expressions, create any XML structure using constructors, and transform XML structures using FLWOR expressions. In addition, XQuery simplifies the tasks encountered when working with namespaces or data types.

Because XML is used to represent and transfer data from a wide variety of sources, XQuery is also widely used for data integration. Even when data is not physically stored as XML, XQuery can be used with any product that provides a processor that creates a logical view of the data as XML. For instance, SOAP

1. In XQuery terminology, the results of a query is an instance of the XQuery data model. We use the term "XML result" for simplicity.

may be used to acquire data from a variety of sources, and XQuery may be used to query the resulting SOAP messages (in XML) together with data found in a relational database (using an XML view).

The XQuery Standard

The XQuery standard is developed by the W3C®, a standards body for the World Wide Web. You can learn more about their work on XQuery here:

<http://www.w3.org/XML/Query.html>

As of January 2007, XQuery 1.0 is the recommended specification of the W3C:

<http://www.w3.org/TR/xquery/>

In December 2008, the W3C published a working draft for the next XQuery version, XQuery 1.1:

<http://www.w3.org/TR/xquery-11/>

Additional information about XQuery can be found at:

<https://www.xquery.com>

Support for the XQuery Standard

Stylus XQuery supports both XQuery 1.0 and XQuery 1.1 as described in [Appendix A “XQuery Support” on page 339](#).

XQuery Examples

This section provides examples of queries – from simple to more complex – to help you become familiar with XQuery.

NOTE: The XQuery examples shown in this section use the database tables and XML files provided with the product in the /examples subdirectory in your Stylus XQuery installation directory.

Example 1: Query Using a FLWOR Expression

The following simple query uses a FLWOR (For each, Let, Where, Order by, Return) expression to return only the rows of the holdings database table that contain a value of AMZN in the stockticker column. In this query, `collection('holdings')` refers to the holdings table in a relational database.

```
for $h in collection('holdings')/holdings
where $h/stockticker='AMZN'
return $h
```

Result

The query returns `$h` directly, so it returns a sequence containing an XML representation of each row.

```
<holdings>
  <userid>Jonathan</userid>
  <stockticker>AMZN</stockticker>
  <shares>3000</shares>
</holdings>
<holdings>
  <userid>Minollo</userid>
  <stockticker>AMZN</stockticker>
  <shares>3000</shares>
</holdings>
```

See [“Data Model Representation of Relational Tables” on page 120](#) for more information about XML views of tables.

Example 2: Creating a Specific XML Structure

In this example, the query returns the same data as Example 1, but it uses an element constructor to create a different XML structure.

```
for $h in collection('holdings')/holdings
where $h/stockticker='AMZN'
return
  <Amazon Client="{ $h/userid}" Shares="{ $h/shares}" />
```

Result

The return clause creates an element named Amazon. It creates two attributes, Client and Shares, which have the values of the userid and shares columns from the relational table.

```
<Amazon Client="Jonathan" Shares="3000" />
<Amazon Client="Minollo" Shares="3000" />
```

Example 3: Combining Data From XML and Relational Sources

Web messages, such as SOAP requests, are XML documents, and they can parameterize or provide data for a query. The following example joins an XML document named request.xml to two relational database tables named holdings and statistical. The request.xml file is joined to the holdings table by the UserId element in the XML file and the userid column of the holdings table. The two tables are joined by the ticker column of the statistical table and the stockticker column of the holdings table.

```
let $request := doc('request.xml')/request
for $user in $request/performance/UserId
return
  <portfolio UserId="{ $user}">
    { $request }
    {
      for $st in collection('holdings')/holdings,
        $stats in collection('statistical')/statistical
      where $st/userid = $user
        and $stats/ticker = $st/stockticker
```

```

        return
        <stock>
            {$stats/companyname}
            {$st/stockticker}
            {$st/shares}
            {$stats/annualrevenues}
        </stock>
    }
</portfolio>

```

Result

The result of this query is an element named `portfolio`. The first child of this element contains the original request from `request.xml`. Subsequently, the query provides the stock information for a given user, obtained from two tables:

```

<portfolio UserId="Jonathan">
  <request>
    <performance>
      <UserId>Jonathan</UserId>
      <start>2003-01-01</start>
      <end>2004-06-01</end>
    </performance>
  </request>
  <stock>
    <companyname>Amazon.com, Inc.</companyname>
    <stockticker>AMZN</stockticker>
    <shares>3000</shares>
    <annualrevenues>7780</annualrevenues>
  </stock>
  <stock>
    <companyname>eBay Inc.</companyname>
    <stockticker>EBAY</stockticker>
    <shares>4000</shares>
    <annualrevenues>22600</annualrevenues>
  </stock>
  <stock>
    <companyname>Int'l Business Machines C</companyname>
    <stockticker>IBM</stockticker>
  </stock>

```

```

    <shares>2500</shares>
    <annualrevenues>128200</annualrevenues>
  </stock>
  <stock>
    <companyname>Progress Software</companyname>
    <stockticker>PRGS</stockticker>
    <shares>23</shares>
    <annualrevenues>493.4</annualrevenues>
  </stock>
</portfolio>

```

Where to Learn More

See [Chapter 4 “Tutorial: Using XQuery” on page 75](#) for a tutorial that focuses on the following major capabilities of XQuery that are fundamental to creating and processing XML:

- Path expressions, which can locate anything in an XML document
- XML constructors, which can create XML documents
- FLWOR expressions (pronounced “flower expressions” and means “for let where order by return”), which allow data to be combined to create new XML structures

What Is XQJ?

The XQuery API for Java (XQJ) is a Java-based API that enables a Java application to submit XQuery queries to an XML data source and process the results. XQJ is designed to support the XQuery language, just as the JDBC API supports the SQL query language. The XQJ standard (JSR 225) is being developed under the Java Community Process. For more information, refer to: <http://www.jcp.org/en/jsr/detail?id=225>

Java Example

The following example illustrates the basic steps that an application performs to execute a query using Stylus XQuery and XQJ. To simplify the example, the code does not include error handling. Multiple Java examples showing how to use XQJ are shipped with the product and are located in the /examples subdirectory in your Stylus XQuery installation directory.

In this example, the application establishes a connection to a relational database using a DDXQDataSource instance. See [“Configuring Connections Explicitly” on page 123](#) for more information about using XQJ to specify connection information for XML and relational data sources.

Example: Executing a Query

```
// import the XQJ classes
import javax.xml.xquery.*;
import com.ddtek.xquery.xqj.DDXQDataSource;

// establish a connection to a data source
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
XQConnection conn = ds.getConnection("myuserid", "mypswd");

// create an expression object that is used to execute a query
XQExpression expr = conn.createExpression();

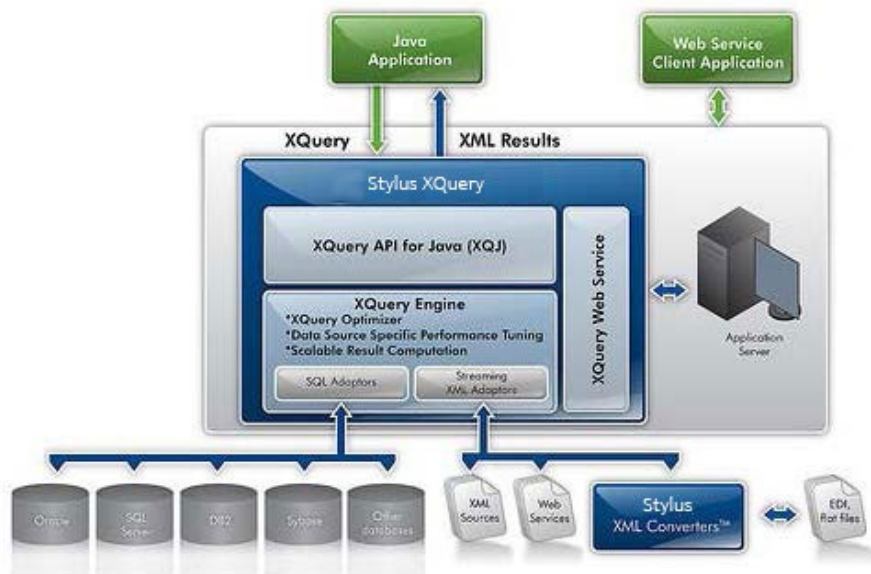
// the query
String es = "for $h in collection('holdings')/holdings " +
            "where $h/stockticker='AMZN' " +
            "return $h";

// execute the query
XQResultSequence result = expr.executeQuery(es);
System.out.println(result.getSequenceAsString(null));
```

```
// free all resources
result.close();
expr.close();
conn.close();
```

Stylus XQuery® Architecture

The following illustration provides a high-level architectural overview of Stylus XQuery.



When you execute a query using Stylus XQuery, Stylus XQuery might process the query in the following fashion:

- 1 A Java application passes a query to the Stylus XQuery implementation of XQJ.
- 2 The XQuery Engine analyzes the query and divides it into one or multiple XQuery expressions to be processed by the adaptors.

- 3 The XQuery Engine sends the query to the SQL adaptor or the Streaming XML adaptor based on its analysis:
 - If a relational source is queried, the XQuery Engine sends the query to the SQL Adaptor. The SQL Adaptor translates the query into SQL, which is used to query the database. The SQL Adaptor receives the results and maps them into XML.
 - If an XML source is queried, the XQuery Engine sends the query to the Streaming XML Adaptor, which executes the query and returns XML results.
 - If a flat or EDI file is queried, the XQuery Engine sends the query to the Streaming XML Adaptor, which relies on the Stylus XML Converters™ to retrieve an XML representation of the flat or EDI file.
- 4 The adaptors send the XML results to the XQuery Engine. If the XML results are obtained from more than one source, the XQuery Engine combines the results.
- 5 The Java application receives results as XML, using XQJ.

Using Stylus XML Converters™

Stylus XML Converters are high-performance Java™ and .NET components that provide bi-directional, programmatic access to virtually any non-XML file including EDI, flat files, and other legacy formats. Stylus XQuery includes several built-in functions that take advantage of the Stylus XML Converters engine to allow you to access as XML data stored in many non-XML formats, including EDI messages, tab-delimited and comma-separated text files, dBASE files, RTF files, and many more.

Using Stylus Studio®

Stylus Studio® is an advanced XML Integrated Development Environment (XML IDE) consisting of hundreds of powerful XML tools in one all-inclusive suite. Among others, Stylus Studio includes for working with XQuery, XSLT, Web services, XML Pipelines, and XML reports.

- The Stylus Studio XQuery editor and XQuery mapper provide integrated support for developing XQuery applications that are powered by Stylus XQuery. The integration is seamless— simply write your code using Stylus Studio's productive XQuery tools as you would normally do. To learn more about Stylus Studio, refer to:
<https://www.stylusstudio.com>

3

Tutorial: Using Stylus XQuery[®]

This tutorial shows how to use Stylus XQuery and XQJ in your Java application to perform tasks that allow you to process queries that access XML and relational data sources, and return XML results. Topics covered in this tutorial include:

- [Specifying the XQuery Version](#)
- [Configuring Connections](#)
- [Executing Queries](#)
- [Querying Data from XML Files or DOM Trees](#)
- [Joining Data from XML and Relational Sources](#)
- [Returning Results with Java XML APIs](#)
- [Preparing XQuery Statements](#)
- [Updating Data in Relational Databases](#)

This tutorial uses the database tables and XML files provided with the product in the /examples subdirectory in the Stylus XQuery installation directory.

In addition to this tutorial, Stylus XQuery is shipped with examples that demonstrate other methods of coding the functionality shown in this tutorial and other Stylus XQuery functionality. These examples are located in the /examples subdirectory in the Stylus XQuery installation directory and explanation of the examples can be found in [Appendix H “Examples” on page 539](#).

Specifying the XQuery Version

Stylus XQuery supports both XQuery 1.0 and XQuery 1.1. You can use a version declaration to indicate whether you want your XQuery code processed using XQuery 1.0 or XQuery 1.1 processing engine. Unless a version is specified explicitly, XQuery code is processed using the XQuery 1.1 processing engine.

Where to Specify Version

You can specify the XQuery version in two places:

- In the main query, using query prolog. Setting the XQuery version in the query prolog affects processing for the entire query, unless a different version has been set for an individual module.
- In one or more modules. Each module can have its own version setting. A module retains its version setting when it is imported.

When to Specify Version

Generally speaking, there is no need to specify the XQuery version because XQuery 1.1 is backwards-compatible with XQuery 1.0.

However, if you write modules that use XQuery 1.1 functionality – to use FLWOR expressions for grouping, for example – consider specifying the version explicitly within that module. Knowing the XQuery version associated with a module can be useful to others who import modules into their XQuery code.

If Stylus XQuery encounters a module with an XQuery 1.0 declaration in the context of a main module whose version is declared (explicitly or not) as 1.1, Stylus XQuery uses its XQuery 1.0 processor for that module and then reverts to using the 1.1 XQuery processor for the remainder of the query.

How to Specify Version

The syntax for the version declaration is:

```
xquery version "[1.0 | 1.1]";
```

Configuring Connections

Stylus XQuery uses `fn:collection()` to access relational data. Stylus XQuery uses XQJ to specify the required database connections and associate the names specified by `fn:collection()` with the database tables.

Using XQJ, you create a connection from an `XQDataSource` instance. The fully qualified class name of the Stylus XQuery `XQDataSource` implementation is:

```
com.ddtek.xquery.xqj.DDXQDataSource
```

The following class provides additional properties for configuring connections to multiple databases:

```
com.ddtek.xquery.xqj.DDXQJDBCCConnection
```

Specifying Connection Information

You can specify connection information to relational data sources using either of the methods shown in the following examples.

Example 1: Using a DDXQDataSource Instance to Specify Connection Information Explicitly

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
```

Example 2: Using the Java Naming and Directory Interface (JNDI)

```
Context ctx = new InitialContext();
DDXQDataSource ds = (DDXQDataSource)ctx.lookup("holdings_ds");
XQConnection conn = ds.getConnection("myuserid", "mypswn");
```

See the [JNDIDataSource](#) example for a complete code sample of how to save and load a Stylus XQuery DDXQDataSource using a JNDI provider.

After specifying connection information using with either DDXQDataSource or JNDI, the `getConnection()` method can be invoked to return an XQJ connection to the database and, optionally, specify the user name and password for the connection:

```
XQConnection conn = ds.getConnection("myuserid", "mypswn");
```

Executing Queries

Next, we create an XQExpression object, which executes an XQuery expression that is read from a file and returns a sequence of results.

First, here is the XQuery expression, `flwor.xq`:

```
for $u in fn:collection('users')/users
```

```

return
  <user>
    <name>{
      $u/firstname,
      $u/lastname
    }</name>
    {
      for $h in collection('holdings')/holdings
      where $h/userid = $u/userid
      return
        <stock>{
          $h/stockticker,
          $h/shares
        }</stock>
    }</user>

```

An XQConnection can create an XQExpression:

```

XQExpression xqExpression = conn.createExpression();
FileReader fileReader = new FileReader("flwor.xq");
XQSequence xqSequence = xqExpression.executeQuery(fileReader);

```

Now that the query results are in a sequence, you can serialize this sequence using the `getSequenceAsString()` method. (Serializing is just one way to handle an XQuery result.)

```
System.out.println(xqSequence.getSequenceAsString());
```

The following result sequence contains a single node, the user element (whitespace has been modified for readability).

```

<user>
  <name>
    <firstname>Jonathan</firstname>
    <lastname>Robie</lastname>
  </name>
  <stock>
    <stockticker>AMZN</stockticker>
    <shares>3000</shares>
  </stock>
  <stock>
    <stockticker>EBAY</stockticker>
  </stock>

```

```
        <shares>4000</shares>
    </stock>
    <stock>
        <stockticker>IBM</stockticker>
        <shares>2500</shares>
    </stock>
    ...
</user>
```

Other similar examples can be found in the [XQJExecute](#) example.

Querying Data from XML Files or DOM Trees

In the previous section, we queried data in a relational database. Now let's query an XML file.

Querying an XML File

Suppose you want to query holdings for a specific customer identified by the `userid` element in a file named `holdings.xml`, which looks like this:

```
<holdings>
  <row>

    <userid>Jonathan</userid>
    <stockticker>AMZN</stockticker>
    <shares>3000</shares>
  </row>
  <row>
    <userid>Minollo</userid>
    <stockticker>EBAY</stockticker>
    <shares>4000</shares>
  </row>
</holdings>
```

Here's an XQuery expression that returns holdings for a customer named Jonathan:

```
doc("holdings.xml")/holdings/row[userid="Jonathan"]
```

Suppose we wanted to return holdings for other customers. If you write an XQuery with an external variable that provides the name of the customer whose holdings you require, the Java application can specify the name of the customer before it executes the query. If you use another external variable to

represent the document, the Java application can pass any document to the query at runtime. For example:

```
declare variable $u as xs:string external;
declare variable $d as document-node(element(*, xs:untyped)) external;
$d/holdings/row[userid=$u]
```

Querying a DOM

Now, let's write Java code to create a DOM tree and bind it to the variable `$d`. Use the following code to create a DOM tree.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);

DocumentBuilder parser = factory.newDocumentBuilder();
File xmlFile = new File("holdings.xml");
Document document = parser.parse(xmlFile);
```

Once you create a DOM tree, you can use XQJ to bind the DOM tree to a variable and query it. First, you create an expression object, and then bind the document to the variable `$d` for this expression.

```
XQConnection conn = ds.getConnection();
XQExpression xqExpression = conn.createExpression();
xqExpression.bindNode(new QName("d"), document);
```

Now, execute the expression and output the result:

```
FileReader fileReader = new FileReader("flwor.xq");

XQSequence xqSequence = xqExpression.executeQuery(fileReader);
System.out.println(xqSequence.getSequenceAsString());
```

Other similar examples can be found in the [ExternalVariables](#) example.

Querying a Directory

You can also query XML files in a directory. See [“Querying Multiple Files in a Directory” on page 288](#) for information about this feature. An example can be found in the [XMLQuery](#) example.

Joining Data from XML and Relational Sources

This tutorial has already explored how XQJ allows XQuery to operate on relational and XML file data stores. Now let's leverage that functionality to query both types of data stores at the same time using a single query.

In this example, we use a Web Service request to provide parameters for the query, and then query a database to create the Web Service response. The Web Service request looks like this:

```
<request>
  <performance>

    <UserId>Jonathan</UserId>
    <start>2003-01-01</start>
    <end>2003-01-01</end>
  </performance>
</request>
```

This request contains only the SOAP message payload. (To simplify the example, the envelope has been omitted.) This request asks for performance data on a user's portfolio within a specific date range.

Now we can compose a query that uses the parameters from the request to create a performance report, which will report the performance of each stock held by each user during the given range.

```

let $request := doc("request.xml")/request
for $user in $request/performance
return
  <portfolio UserID="{ $user/UserId}">
    { $request }
    {
      for $h in collection("holdings")/holdings
      where $h/userid = $user/UserId
      return
        <stock>
          {
            $h/stockticker,
            $h/shares
          }
        </stock>
    }
  </portfolio>

```

First, establish a connection to the data source.

```
XQConnection conn = ds.getConnection();
```

Create an XQExpression object that executes the XQuery expression and returns a sequence of results.

```

FileReader fileReader = new FileReader("flwor.xq");
XQExpression xqExpression = conn.createExpression();
XQSequence xqSequence = xqExpression.executeQuery(fileReader);

```

With the query results in a sequence, serialize this sequence using the `getSequenceAsString()` method.

```
System.out.println(xqSequence.getSequenceAsString());
```

The result looks like this (whitespace has been modified for readability):

```

<portfolio UserID="Jonathan">
  <request>
    <performance>
      <UserId>Jonathan</UserId>
      <start>2003-01-01</start>

```

```

        <end>2004-06-01</end>
    </performance>
</request>
    <stock>
        <stockticker>PRGS</stockticker>
        <shares>23</shares>
    </stock>
    <stock>
        <stockticker>AMZN</stockticker>
        <shares>3000</shares>
    </stock>
    <stock>
        <stockticker>EBAY</stockticker>
        <shares>4000</shares>
    </stock>
    <stock>
        <stockticker>IBM</stockticker>
        <shares>2500</shares>
        <shares>2500</shares>
    </stock>
</portfolio>

```

Other similar examples can be found in the [XQJExecute](#) example.

Returning Results with Java XML APIs

Often, applications need to retrieve XQuery results as DOM, SAX, or StAX. XQSequence, as shown previously in this tutorial, allows access to the result as a direct mapping of the XQuery sequence. Within an XQSequence, XQItem objects represent each item in an XQuery sequence.

NOTE: Instantiating each item in an XQItem object affects performance because it requires the processing to create multiple objects. Use XQItem object judiciously.

Next, we'll show you how to process an XQuery sequence and return the output as DOM, SAX, or StAX.

First, create an XQExpression object that executes the XQuery expression and returns a sequence of results:

```
DDXQDataSource ds = new DDXQDataSource();
...
XQConnection conn = ds.getConnection("myuserid", "mypswd");
FileReader fileReader = new FileReader("flwor.xq");
XQExpression xqExpression = conn.createExpression();
XQSequence xqSequence = xqExpression.executeQuery(fileReader);
```

DOM

To return the output from a result sequence as a DOM tree, we iterate over each DOM node in the XQuery sequence to extract the DOM content and print the DOM node to the standard `System.out`. For example, if you have J2SE 1.4.x, use the following code, which assumes all items in the result sequence are node items:

```
while(xqSequence.next()){
    Node domNode = xqSequence.getNode();
    System.out.println(domNode);
}
```

If you have J2SE 1.5 and higher, the method is different; it is shown in the [ResultRetrieval](#) example.

SAX

To return the output from a result sequence as a SAX event stream rather than a string, create a SAX event handler (named `SimpleSAXEventHandler`, in this case) that sends the results to the standard `System.out` as shown in the following code:

```
SimpleSAXEventHandler anEventHandler = new SimpleSAXEventHandler(System.out);
xqSequence.writeSequenceToSAX(anEventHandler);
```

The complete application can be found in the [ResultRetrieval](#) example.

StAX

To return the output from a result sequence as a StAX event stream rather than as a string, create a StAX reader as shown in the following code:

```
XMLStreamReader reader = xqSequence.getSequenceAsStream();
```

You can use this StAX reader functionality like any other StAX stream reader. For example, the following code reads one event at a time and prints the event type together with the associated event names.

```
private static void formatOutput(XMLStreamReader reader) throws
XMLStreamException {
while(true) {
    int event = reader.next();
    if(event == XMLStreamConstants.END_DOCUMENT) {
        return;
    }
    switch (event) {
        case XMLStreamConstants.START_ELEMENT:
            System.out.println("Start tag: ");
            printNames(reader);
            break;

        case XMLStreamConstants.END_ELEMENT:
```

```

        System.out.println("End tag");
        printNames(reader);
        break;

    case XMLStreamConstants.CHARACTERS:
        System.out.println("Text");
        printChars(reader);
        break;
    }
}
...

```

Other similar examples can be found in the [ResultRetrieval](#) example.

Preparing XQuery Statements

Typically, when a query is executed, the query is parsed and optimized before it is run. To avoid incurring this overhead each time the query is used, you can prepare the same query once and execute it multiple times.

The following is the code for creating a prepared query. Only the last line differs from the code used to create a query in our example in [“Querying Data from XML Files or DOM Trees” on page 65](#).

```

DDXQDataSource ds = new DDXQDataSource();
XQConnection conn = ds.getConnection();

FileReader fileReader = new FileReader("flwor.xq");

XQPreparedExpression preparedExpression = conn.prepareExpression(fileReader);

```


Once the query is prepared, use an `executeQuery()` call to execute it.

```
XQSequence xqSequence = preparedExpression.executeQuery();
System.out.println(xqSequence.getSequenceAsString(null));
```

Queries can accept parameters that can be changed between executions. For example, you may want to prepare a query that selects holdings based on a particular customer. In the following query, the value of `userid` changes each time this XQuery is run. (Each `userid` is associated with a specific customer.)

```
declare variable $l as xs:string external;
collection("holdings")/holdings[userid=$l]
```

The value of `$l` is set using XQJ. Let's run this twice, each time for different users.

```
preparedExpression.bindString(new QName("l"), "Jonathan");
xqSequence xqSequence = preparedExpression.executeQuery();
System.out.println("\n\nHoldings for Jonathan:\n\n");
System.out.println(xqSequence.getSequenceAsString(null));
```

```
preparedExpression.bindString(new QName("l"), "Minollo");
xqSequence xqSequence = preparedExpression.executeQuery();
System.out.println("\n\nHoldings for Minollo:\n\n");
System.out.println(xqSequence.getSequenceAsString(null));
```

Other similar examples can be found in the [ResultRetrieval](#) example.

Updating Data in Relational Databases

You can execute updating expressions using either `XQExpression` or `XQPreparedStatement` objects. The result of an updating query is always an empty sequence.

The following example executes an updating expression in XQJ using an `XQExpression` object. The updating expression inserts data into the holdings database table.

```
// import the XQJ classes
import javax.xml.xquery.*;
import com.ddtek.xquery.xqj.DDXQDataSource;

// establish a connection to a relational data source
// specify the URL and the user ID and password
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
XQConnection conn = ds.getConnection("myuserid", "mypswn");
// create an expression object that is used to execute a query
XQExpression xqExpression = conn.createExpression();

// the query
String es = "ddtek:sql-insert('holdings'," +
            "'userid','Minollo','stockticker','TIVO','shares',200)";

// execute the query
XQResultSequence result = xqExpression.executeQuery(es);

// free all resources
result.close();
xqExpression.close();
conn.close();
```

Other examples can be found in the [RDBMSUpdate](#) example.

See “[Updating Relational Data](#)” on page 267 for more information about updating data in relational databases.

4 Tutorial: Using XQuery

W3C defines the XML Query (XQuery) language for querying XML and combining data from documents, databases, Web pages, and other sources. Some common use cases for XQuery involve XML publishing to create XML for Web messages, dynamic web sites, and publishing applications. The source for query data might be found in XML files, relational databases, legacy files such as EDI, or from multiple combined sources.

Some of the queries in this tutorial operate on XML stored in files, some on an XML view of a relational database, and some work on both. All of the examples in this tutorial have been tested with Stylus XQuery. Because not all XQuery implementations access relational data in the same way, this tutorial uses `fn:collection()`, which Stylus XQuery uses to access relational tables.

Most XQuery functionality, such as arithmetic operators, comparisons, function calls, and functions, is familiar to most programmers. This tutorial focuses on the three major capabilities that distinguish XQuery, each of which is fundamental to processing and creating XML:

- Path expressions, which can locate anything in an XML document.
- XML constructors, which can create any XML document.
- FLWOR expressions (pronounced “flower expressions”), which allow data to be combined to create new XML structures. They are similar to SQL Select statements that have From and Where clauses, but are adapted for XML processing.

Together, these capabilities make XQuery easier to use than other languages when processing and creating XML using data from XML or relational sources.

This chapter covers the following subjects:

- [Finding XML Nodes: Path Expressions](#)
- [Creating XML: XML Constructors](#)
- [Restructuring Data: FLWOR Expressions](#)
- [Grouping Data](#)
- [Summary](#)

Finding XML Nodes: Path Expressions

Just as SQL needs to be able to access any row or column in a relational table, XQuery needs to be able to access any node in an XML document. XML structures have both hierarchy and sequence, and can be quite complex. Path expressions directly support XML hierarchy and sequence, and allow you to navigate any XML structure.

In this section, we discuss path expressions using an XML document, and then show path expressions used on an XML view of a relational table.

Path Expressions for XML Sources

Let's explore path expressions using the following XML document, `portfolio.xml`, which consists of a `portfolio` element with `name` and `stocks` subelements.

```
<?xml version="1.0"?>
<portfolio id="Jonathan">
```

```
<name>  
  <first>Jonathan</first>  
  <last>Robie</last>  
</name>  
<stocks>  
  <stock>  
    <ticker>AMZN</ticker>  
    <shares>3000</shares>  
  </stock>
```

```

    <stock>
      <ticker>EBAY</ticker>
      <shares>4000</shares>
    </stock>
    <stock>
      <ticker>IBM</ticker>
      <shares>2500</shares>
    </stock>
    <stock>
      <ticker>PRGS</ticker>
      <shares>23</shares>
    </stock>
  </stocks>
</portfolio>

```

`fn:doc()` returns a document. The following example shows how to use `fn:doc()` with an absolute URI.

```
doc("file:///c:/data/xml/portfolio.xml")
```

The following example shows how to use `fn:doc()` with a relative URI.

```
doc("portfolio.xml")
```

By setting the Base URI, you can set the directory that is used to resolve relative URIs.

```
declare base-uri "file:///c:/data/xml/";
doc("portfolio.xml")
```

A path expression consists of a series of one or more “steps”, separated by a slash (/) or double slash (/). Every step evaluates to a sequence of nodes. For example, consider the expression:

```
doc("portfolio.xml")/portfolio/name
```

The first step, `doc("portfolio.xml")`, returns a document node that represents the portfolio document.

The second step, `portfolio`, is a name test that specifies the name of an element; it returns the portfolio element at the top of the document, which is a child of the document node.

The third step, `name`, returns the element named “name”, which is a child of the portfolio element.

Result of the Query Expression

```
<name>
  <first>Jonathan</first>
  <last>Robie</last>
</name>
```

If a name test is preceded by the `@` character, the name test matches an *attribute* rather than an element. For example, the expression `doc("portfolio.xml")/portfolio/@id` returns the id attribute of the portfolio element.

The double slash (`//`) allows steps to operate on any descendant of a node. For example, the expression `doc("portfolio.xml")//name` matches any element named name, anywhere in the portfolio document.

Predicates

A *predicate* can be added to a step to set conditions for matching nodes. Predicates often set a condition on the children of a node. For example, the following path matches stock elements that contain a ticker element with the value “AMZN”.

```
doc("portfolio.xml")//stock[ticker='AMZN']
```

Using the sample data, this expression produces the following result:

```
<stock>
  <ticker>AMZN</ticker>
  <shares>3000</shares>
</stock>
```

Conditions in a predicate can be combined using “and” and “or”, as in the following expression.

```
doc("portfolio.xml")//stock[ticker='AMZN' or ticker='EBAY']
```

Conditions can be negated using `fn:not()`; for example, the following expression matches stock elements that do not have a ticker element with the value “AMZN”:

```
doc("portfolio.xml")//stock[not(ticker='AMZN')]
```

One type of predicate is a *numeric* predicate, which sets a condition on the position of a node in a sequence. For example, the following expression finds the first stock element in a portfolio.

```
doc("portfolio.xml")//stocks/stock[1]
```

To understand how numeric predicates work in XQuery, you must know how XQuery evaluates a slash (/), as described in the following steps:

- 1 The expression on the left side of a slash is evaluated to produce a sequence of nodes.
- 2 The expression on the right side is evaluated for each context node drawn from the expression on the left side, and the results are combined.
- 3 When the numeric predicate is evaluated, it is evaluated for a given context node.

For example, in the preceding expression, when the numeric predicate is evaluated, the context node is a `stocks` element, the

name test stock evaluates to a sequence of stock elements, and the numeric predicate matches the first stock in this sequence.

The following expression matches the first ticker element on each stock element:

```
doc("portfolio.xml")//stock/ticker[1]
```

To get the first ticker element in the document, use parentheses to make the expression on the left of the numeric predicate evaluate to the sequence of all ticker elements in the document:

```
(doc("portfolio.xml")//stock/ticker)[1]
```

Path Expressions for Relational Sources

When XQuery is used to query relational data, relational tables are treated as though they are XML documents, and path expressions work the same way as they do for XML. Because relational tables have a simple structure, path expressions used for tables are usually simple.

Each XQuery implementation has its own way of accessing a relational table. Stylus XQuery uses the `fn:collection()` to access a relational table. For example, the following expression accesses the holdings table:

```
collection('holdings')
```

Each XQuery implementation must also decide how to map relational tables into XML in the XML view. The SQL 2003 standard has defined a standard set of mappings for this purpose as part of SQL/XML.

Here is a SQL/XML mapping of the holdings table; this mapping represents each row as a holdings element, and represents each column of the table (userid, stockticker, shares) as an element that is a child of the holdings element:

```
<holdings>
```

```

    <userid>Jonathan</userid>
    <stockticker>AMZN</stockticker>
    <shares>3000</shares>
</holdings>
...
<holdings>
  <userid>Minollo</userid>
  <stockticker>AMZN</stockticker>
  <shares>3000</shares>
</holdings>
...

```

Once you understand the structure of the XML view, you can easily see how path expressions are applied to it. For example, the following expression finds holdings for the user whose `userid` is “Minollo”.

```
collection('holdings')/holdings[userid='Minollo']
```

Stylus XQuery Speaks SQL

Because relational data is queried as if it were XML, you might think that relational tables are actually extracted from the database, turned into XML documents, and then queried, but this would be very inefficient.

To the user, Stylus XQuery makes all data look like XML, but to a SQL database, the implementation speaks SQL. Before evaluating the preceding expression, for example, Stylus XQuery converts it to a SQL expression similar to this one:

```

SELECT userid, stockticker, shares
FROM holdings
WHERE userid='Minollo'

```

Creating XML: XML Constructors

Now that we have seen how to locate anything in an XML document or a relational table, let's learn how to create new XML structures using XML constructors.

Literal XML constructors

The most simple constructors are *literal XML constructors*, which use the same syntax as XML. For example, the following XML text is also an XQuery expression that creates the equivalent XML structure.

```
<stock role='eg'>
  <ticker>AMZN</ticker>
  <shares>3000</shares>
</stock>
```

This example uses only elements and attributes, but processing instructions, comments, and CDATA sections can also be used in XML constructors.

Enclosed Expressions

In literal XML constructors, you can use curly braces ({ }) to add content that is computed when the query is run. This is called an *enclosed expression*. For example, the following expression creates a date element whose content is the current date, which is computed when the query is run:

```
<date>{ current-date() }</date>
```

The result is an element named date with the current date.

To see why enclosed expressions are necessary, consider the following expression:

```
<date> current-date() </date>
```

This expression evaluates to the following XML element:

```
<date> current-date() </date>
```

Path expressions are frequently used in enclosed expressions. The following expression creates a portfolio element for Minollo, and then extracts Minollo's holdings from the holdings table:

```
<portfolio name='Minollo'>
  { collection('holdings')/holdings[userid='Minollo'] }
</portfolio>
```

Restructuring Data: FLWOR Expressions

The XQuery FLWOR expression is similar to a SQL Select statement that has From and Where clauses. FLWOR is pronounced "flower," and is an acronym for the keywords used to introduce each clause (for, let, where, order by, and return).

Here is a FLWOR expression that returns holdings for AMZN:

```
for $h in collection('holdings')/holdings
where $h/stockticker = 'AMZN'
order by $h/shares
return $h
```

In the preceding query, the FLWOR expression performs the following functions:

- The for clause binds the variable \$h to each holdings element.
- The where clause filters out bindings of \$h for which the stockticker element does not contain the value AMZN.
- The order by clause determines the order.

- The return clause produces a result for each binding of \$h.

FLWOR expressions are frequently used to combine related information. The possible combinations are generated by using variables in the for clause and using a where clause to filter out combinations that are not useful. This is known as a "join". Consider the following expression:

```
for $u in collection('users')/users,
    $h in collection('holdings')/holdings
where $u/userid=$h/userid
order by $u/lastname, $u/lastname
return
    <holding>
    {
        $u/firstname,
        $u/lastname,
        $h/stockticker,
        $h/shares
    }
    </holding>
```

This expression finds every pair of users elements and holdings elements whose userid child element has the same value, and then builds a holding element that describes the user and his holdings.

Now, let's look at a FLWOR expression that uses a let clause:

```
let $h := collection('holdings')/holdings
return count($h)
```

A let clause binds a variable to a sequence, which often contains more than one item. In the preceding query, \$h is bound to all of the holdings elements in the collection, and the return clause is evaluated. Note the difference between a for clause and a let clause: a for clause always iterates over a sequence, binding a variable to each item; a let clause simply binds a variable to the entire sequence.

In the preceding expression, the result is 8. In contrast, if you use the following for clause:

```
for $h in collection('holdings')/holdings
return count($h)
```

The result is a sequence of eight numbers: 1 1 1 1 1 1 1 1.

In some cases, you might find it useful to combine for and let clauses. In the following expression, these two clauses are combined to produce a result that counts the number of stock holdings for each user.

```
for $u in collection('users')/users
let $h := collection('holdings')/holdings[userid=$u/userid]
order by $u/lastname, $u/firstname

return
  <user nstocks="{count($h)}">
    {
      $u/firstname,
      $u/lastname
    }
  </user>
```

XML Reporting for Relational Sources

Many applications need to create rich XML structures from relational sources. For example, Web sites generally create hierarchical displays of the data found in a relational database, and Web messages are often very complex hierarchical structures. For these applications, XQuery can act as an “XML report writer.”

The database tables used in this section are as follows:

■ **users Table**

userid	firstname	lastname	othername
Minollo	Carlo	Innocenti	
Jonathan	Jonathan	Robie	William

■ **holdings Table**

userid	stockticker	shares
Jonathan	PRGS	23
Minollo	PRGS	4000000
...		

■ **statistical Table**

id	companyname	ticker	percentagechange	annualrevenues	location
1	Apple Computer, Inc.	AAPL	-40.80%	5250	Cupertino
2	Accrue Software, Inc.	ACRU	-57.60%	4.21	Freemont
...					

This query creates a portfolio for each user:

```
<portfolios>
{
  for $u in collection('users')/users
  order by $u/userid
  return
    <portfolio id="{ $u/userid }">
      <name>
        <first>{data($u/firstname)}</first>
        <last>{data($u/lastname)}</last>
      </name>
      <stocks>
        {
          for $h in collection('holdings')/holdings
          where $h/userid = $u/userid
          order by $h/stockticker
          return
            <stock>
              <ticker>{data($h/stockticker)}</ticker>
              <shares>{data($h/shares)}</shares>
            </stock>
        }
      </stocks>
    </portfolio>
}
</portfolios>
```


Here is the result of the preceding query.

```
<portfolios>
  <portfolio id="Jonathan">
    <name>
      <first>Jonathan</first>
      <last>Robie</last>
    </name>
    <stocks>
      <stock>
        <ticker>AMZN</ticker>
        <shares>3000</shares>
      </stock>
      <stock>
        <ticker>EBAY</ticker>
        <shares>4000</shares>
      </stock>
      <stock>
        <ticker>IBM</ticker>
        <shares>2500</shares>
      </stock>
      <stock>
        <ticker>PRGS</ticker>
        <shares>23</shares>
      </stock>
    </stocks>
  </portfolio>
```

```

<portfolio id="Minollo">
  <name>
    <first>Carlo</first>
    <last>Innocenti</last>
  </name>
  <stocks>
    <stock>
      <ticker>AMZN</ticker>
      <shares>3000</shares>
    </stock>
    <stock>
      <ticker>EBAY</ticker>
      <shares>4000</shares>
    </stock>
    <stock>
      <ticker>LU</ticker>
      <shares>40000</shares>
    </stock>
    <stock>
      <ticker>PRGS</ticker>
      <shares>4000000</shares>
    </stock>
  </stocks>
</portfolio>
</portfolios>

```

NOTE: The query that created this XML result uses the data function, which returns only the value of the stockticker column. Without the data function, the value would be surrounded with an element named stockticker, resulting in, for example:

```

<ticker>
  <stockticker>AMZN</stockticker>
</ticker>

```

Processing XML and Relational Together

In some applications, you may need to use XML and relational data together. For example, a configuration file or an incoming Web message might provide information needed to parameterize a query. Suppose you have an XML file that contains a request for a particular kind of report, and your query is to produce that report. For example, the following XML file, `request.xml`, contains a request to show the performance of Jonathan's stocks during the period from 2003-01-01 to 2004-06-01.

```
<?xml version="1.0"?>
<request>
  <performance>
    <UserId>Jonathan</UserId>
    <start>2003-01-01</start>
    <end>2004-06-01</end>
  </performance>
</request>
```

Here is a query that creates a portfolio for the user specified in a request file, during the requested period:

```
declare base-uri "file:///c:/programs/examples/JoinXMLToRelational/";
declare variable $request := doc('request.xml')/request;

for $user in $request/performance/UserId,
  $start in $request/performance/start,
  $end in $request/performance/end
return
  <portfolio UserId="{ $user }">
    { $request }
    {
      for $st in collection('holdings')/holdings,
        $stats in collection('statistical')/statistical
      where $st/userid = $user
        and $stats/ticker = $st/stockticker
      return
```

```

<stock>
  { $stats/companyname }
  { $st/stockticker }
  { $st/shares }
  { $stats/annualrevenues }
  {
    let $hist :=
      for $h in collection('historical')/historical
      where $h/ticker = $st/stockticker
        and xs:date($h/datetraded) gt xs:date($start)
        and xs:date($h/datetraded) lt xs:date($end)
      return $h
    return
      <performance>
        <min>{min($hist/adjustedclose)}</min>
        <max>{max($hist/adjustedclose)}</max>
        <daily>
          {
            for $h in $hist
            return <day>{$h/datetraded, $h/adjustedclose}</day>
          }
        </daily>
      </performance>
  }
</stock>
}
</portfolio>

```

Grouping Data

The previous topic, [Restructuring Data: FLWOR Expressions](#), described how to use XQuery FLWOR expressions with XML and relational data sources to restructure data. This topic describes how to group data using the XQuery FLWOR expression window clause.

1.1 The XQuery FLWOR expression window clause is supported in XQuery 1.1 only.

This section covers the following topics:

- [What Is Grouping](#)
- [The window Clause](#)
- [Example: Tumbling Windows](#)
- [Example: Positional Grouping](#)
- [Example: Sliding Windows](#)

What Is Grouping

Grouping is a technique that allows you to group XML data and then perform some sort of query – a transformation, for example – on the data in that group.

In XQuery 1.1, you can achieve grouping using the window clause in a FLWOR expression. The window clause is powerful because it allows you to bind the for clause variable to a group of elements, instead of to a single element only. In XQuery 1.0, the for clause variable in FLWOR expressions could be bound to a single element only.

The window Clause

The *window clause* in the XQuery FLWOR expression allows you to group data in sequences of consecutive items; these sequences are called *windows*. The start and end of a window are based on user-defined criteria – the WindowStartCondition (start \$var when ExprSingle) and WindowEndCondition (end \$var when ExprSingle), respectively. To create the window, the window clause iterates over the sequence, referred to as the *binding sequence*. The resulting window contains the binding sequence's start item, end item, and all the items in between.

In a window clause, the starting item of the window is determined by the window type.

Types of Windows

You can use the window clause to create two types of windows:

- **Tumbling** – tumbling windows are defined as windows whose items never overlap. The item that starts one window always follows the last item of the previous window. Thus, no two windows drawn from the same binding sequence can contain the same items.
- **Sliding** – sliding windows, on the other hand, are defined as windows that can overlap. That is, one window might contain the same item as another window drawn from the same binding sequence. This can occur because every item in the binding sequence that makes the `WindowStartCondition` true is the starting item for each new window.

Examples of both types of windows appear in the following sections.

Example: Tumbling Windows

Consider the following XML document, which contains information customer orders. Note that the document structure is flat – customer and order elements are intermingled:

```
<?xml version="1.0"?>
<orders>
  <customer id="1"/>
  <order type="book" id="1" price="10.0"/>
  <order type="DVD" id="3" price="24.0"/>
  <customer id="2"/>
  <order type="game" id="5" price="50.0"/>
</orders>
```

We want to use XQuery to group all orders by customer, like this:

```
<?xml version="1.0"?>
<orders>
  <customer id="1">
    <order type="book" id="1" price="10.0"/>
    <order type="DVD" id="3" price="24.0"/>
  </customer>
  <customer id="2">
    <order type="game" id="5" price="50.0"/>
  </customer>
</orders>
```

Using the tumbling windows in XQuery 1.1, the code required to generate the same output is straightforward and concise. Here, the XQuery 1.1 code iterates through all of the elements in the XML document. The start of the binding sequence is an element *customer*; the end of the binding sequence occurs when the element immediately after the context is not an *order* (that is, in this example, it is another *customer*).

```
<orders>{
  for tumbling window $customer in $data/orders/*
  start $start when $start/self::customer
  end next $next when $next[self::customer]
  return
    <customer>{
      $start/@*,
      subsequence($customer, 2)
    }</customer>
}</orders>
```

While generating this output is possible in XQuery 1.0, the recursive function required to process one sibling at a time is somewhat complex and not especially efficient, as shown here:

```
declare function local:orders($c as element(*, xs:untyped) )
  as element()*
{
  local:next($c/following-sibling::*[1])
};
```

```

declare function local:next($o as element(*, xs:untyped)* )
  as element()* *
{
  if($o/self::order) then
    ($o, local:next($o/following-sibling::*[1][self::order]))
  else ()
};

<orders>{
for $ele in $data/orders/customer
return
  <customer>{
    $ele/@*,
    local:orders($ele)
  }</customer>
}</orders>

```


Example: Positional Grouping

The XQuery 1.1 window clause also allows you to define grouping criteria based on position within the XML source. Imagine, for example, an XML document with the following structure containing thousands of *book* elements:

```
<?xml version="1.0"?>
<books name="My books">
  <book bookid="1" pubdate="03/01/2002">
    <title>Java Web Services</title>
    <authors>
      <author>David A. Chappel</author>
      <author>Tyler Jewell</author>
    </authors>
    <subject>Web Services</subject>
  </book>
  ...
</books>
```

The XQuery code required to split a single document into multiple documents of, say, 10 *book* elements each might look like this:

```
declare variable $window_size = 10;

for tumbling window $customer_orders in $books/books/book
start when true()
end at $i when $i mod $window_size = 0
return
  serialize-to-url(
    <books>{$customer_orders}</books>
    concat("books", $end_pos div $window_size, ".xml", ""))
```

Example: Sliding Windows

The previous examples showed how to group data using tumbling windows – that is, windows with adjacent items. Sliding windows contain items that can overlap.

Consider this example from the [XQuery 1.1 W3C Working Draft 3 December 2008](#). We have a document that contains a list of colors, like this:

```
<?xml version="1.0"?>
<doc>
  <data>Green</data>
  <data>Pink</data>
  <data>Lilac</data>
  <data>Turquoise</data>
  <data>Peach</data>
  <data>Opal</data>
  <data>Champagne</data>
</doc>
```

Using a FLWOR expression with a sliding window clause:

```
declare option ddtek:serialize "indent=yes";
<root>{
  for sliding window $item in doc("arrange_rows.xml")/*/data
    start at $sp when true()
    end at $sep when $sep - $sp = 2
  return <window>{$item}</window>
}</root>
```

We can generate a sequence of items, grouped by three, each successive group containing two items overlapping those in the previous group:

```
<?xml version="1.0"?>
<root>
  <window>
    <data>Green</data>
    <data>Pink</data>
    <data>Lilac</data>
```

```

</window>
<window>
  <data>Pink</data>
  <data>Lilac</data>
  <data>Turquoise</data>
</window>
<window>
  <data>Lilac</data>
  <data>Turquoise</data>
  <data>Peach</data>
</window>
<window>
  <data>Turquoise</data>
  <data>Peach</data>
  <data>Opal</data>
</window>
<window>
  <data>Peach</data>
  <data>Opal</data>
  <data>Champagne</data>
</window>
<window>
  <data>Opal</data>
  <data>Champagne</data>
</window>
<window>
  <data>Champagne</data>
</window>
</root>

```

Summary

The major capabilities of XQuery that distinguish it from other programming languages are its ability to:

- Locate any content in an XML document
- Create any XML document
- Combine data to create new XML structures using FLWOR expressions

For a more extensive XQuery tutorial, refer to *XQuery: A Guided Tour* at:

<https://www.stylusstudio.com/tutorials/xquery-tutorials.html>

Stylus XQuery, an implementation of XQuery, allows you to query both relational and XML sources and combine the data into one result.

5 Tutorial: The XQuery Update Facility

Stylus XQuery supports the XQuery Update Facility 1.0 (XUF). XUF is an extension of the XQuery language that allows making changes to data that are manipulated inside the XQuery.

This chapter describes how Stylus XQuery supports XUF and provides examples for its use. For more details about the XUF specification, see the W3C Candidate Recommendation here:

<http://www.w3.org/TR/2008/CR-xquery-update-10-20080801/>

This chapter covers the following topics:

- “Support Overview”
- “XUF Examples”
- “Storing Query Results”
- “Replacing Node Values”
- “Inserting a New Node”
- “Renaming a Node”
- “Transforming Query Results”
- “Updating Data Sources”

Support Overview

Stylus XQuery supports the complete XUF specification for

- Individual XML documents
- XML streams
- File collections

Updating Relational Data

Stylus XQuery does not support XUF for relational data sources. XUF expressions can operate on nodes containing data that originates from a relational data source, but these changes are not propagated to the relational data source. Consider this example:

```
let $myName := <name>{collection("USERS")/USERS[USERID="02"]  
                  /FIRSTNAME/text()}</name>
```

Here, you can use XUF to modify `$myName`, but you cannot change values from the `collection()` function because those nodes are retrieved directly from a relational data source.

You can, however, use the `ddtek:sql-insert/update/delete` functions to perform update operations directly against an RDBMS. See [“Updating Relational Data” on page 267](#) for more information.

XUF Expressions

The XUF specification introduces several new expressions as extensions to the XQuery specification:

- Insert
- Delete
- Replace

- Rename
- Transform

These and other expressions and functions that support XUF are described and illustrated in the examples in the sections that follow.

XUF Examples

Stylus XQuery bundles several examples that help you understand how to use XUF in your XQuery applications. These examples are located in the \examples\UpdateFacility folder where you install Stylus XQuery.

These examples, and the XUF expressions and functions they illustrate, are described in the sections that follow this one.

Sample Files

Most Stylus XQuery XUF examples use these sample files:

- holdings.xml – an XML document that lists stock holding information. The holdings.xml document has the following structure:

```
<table name="HOLDINGS">
  <holdings>
    <UserId>Jonathan</UserId>
    <StockTicker>PRGS</StockTicker>
    <Shares>23</Shares>
  </holdings>
  <holdings>
    <UserId>Minollo</UserId>
    <StockTicker>PRGS</StockTicker>
    <Shares>4000000</Shares>
  </holdings>
```

```
...
</table>
```

- **users.xml** – an XML document that lists individuals' information. The **users.xml** document has the following structure:

```
<table name="users">
  <users>
    <UserId>Minollo</UserId>
    <FirstName>Carlo</FirstName>
    <LastName>Innocenti</LastName>
    <OtherName/>
    <MemberSince>2004-06-16T00:00:00</MemberSince>
  </users>
  ...
</table>
```

Storing Query Results

The XUF specification defines the `fn:put()` function as

```
fn:put($node as node(), $uri as xs:string) as empty-sequence()
```

where:

- `$node` can be any node, whether or not it is updated
- `$uri` can be any valid URI and is processed in the same way as a URI provided to the `ddtek:serialize-to-url()` function

The `fn:put()` function is similar to the `ddtek:serialize-to-url()` function, but it is executed at the end of the XQuery execution, when all pending updates have been applied – but before control is returned to the calling application. See [“ddtek:serialize-to-url” on page 420](#) for more information on that function.

Example

The following code shows an example of the `fn:put()` function being used in an updating XQuery:

```
declare variable $holding := doc('holdings.xml');  
delete nodes $holding/table/holdings[UserId = "Minollo"],  
put($holding, "newHolding.xml")
```

Replacing Node Values

The `replace` expression is an updating expression that can be used to replace a node or a node's value. Its syntax varies based on the type of replacement.

Example

The `change-values.xq` example queries `holdings.xml`. It looks for a user, Minollo (`/table/holdings[UserId eq 'Minollo']`), and increases his stock holdings (`<shares>`) by 20%, replacing the current value with the calculated value.

Results of the query are written to a new XML document, `more-shares.xml`, using the `fn:put()` function:

```
for $holding in doc("/examples/xml/holdings.xml")
    /table/holdings[UserId eq 'Minollo']
return
replace value of node $holding/Shares with xs:integer($holding/Shares * 1.2),
    put(doc("/examples/xml/holdings.xml"), "/examples/xml/mo-shares.xml")
```

Inserting a New Node

The `insert` expression allows you to insert new nodes into XML documents. The XUF specification defines the `insert` expression as

```
insert (node|nodes) target
```

where:

- `(node|nodes)` can be one or more individual XML nodes; you can use either word regardless of the number of nodes being inserted.
- `target` is the target expression. You use it both to identify into what you want the node inserted (an XML document, for example) and where. New nodes can be inserted at the start of the target (as `first`), and the end (as `last`), or after or before any node you specify.

Note that if you do not specify placement in the target expression, Stylus XQuery inserts new nodes at the end of the target in document order.

Example

In this example, for every user in `users.xml`, `insert-nodes.xq` adds 1000 shares of DDTEK stock to that user's holdings. It creates a new `<holdings>` node for those users who do not already have DDTEK – or who are not already listed in the `holdings.xml` document – and writes the result to a new XML document. Note that the `insert node` expression specifies placement (as last) within the target document.

```
for $user in doc("/examples/xml/users.xml")/table/users
let $ddtekShares := doc("/examples/xml/holdings.xml")/table/holdings[UserId
eq $user/UserId and StockTicker eq "DDTEK"]
return
  if( $ddtekShares ) then
    replace value of node $ddtekShares/Shares with $ddtekShares/Shares + 1000
  else
    insert node
      <holdings>
        <UserId>{$user/UserId/text()}</UserId>
        <StockTicker>DDTEK</StockTicker>
        <Shares>1000</Shares>
      </holdings>
    as last into doc("/examples/xml/holdings.xml")/table,
  put(doc("/examples/xml/holdings.xml"), "/examples/xml/more-ddtek-
    holdings.xml")
```

Renaming a Node

The `rename` expression allows you to replace the name property for a specific node. Attributes and descendants of the specified node are not affected by the `rename` operation.

Example

The `rename-nodes.xq` XQuery changes an XML document by renaming `<UserID>` node to `<ID>` and saving the result to a new document using the `fn:put()` function.

```
for $user in doc("../xml/users.xml")/table/users
return
    rename node $user/UserId as QName("", "ID"),
    put(doc("../xml/users.xml"), "new_users.xml")
```

Example – Using XQJ

You can execute XQuery programmatically using XQJ. In this example, a Java application is used to execute XQuery that uses the same `rename` and `put` XUF expressions to rename nodes in an XML document and create a new XML document with the renamed nodes:

```
// import the XQJ classes
import javax.xml.xquery.*;
import com.ddtek.xquery.xqj.DDXQDataSource;
import com.ddtek.xquery.xqj.DDXQJDBCConnection;

public class XUF {

    public static void main(String[] args) throws Exception {

        XQConnection xqconnection = null;
        XQPreparedExpression xqExpr = null;
```

```

try {
    DDXQDataSource dataSource = new DDXQDataSource();
    xqconnection = dataSource.getConnection();

    // the query
    String xquery =
        "for $user in doc('../xml/users.xml')/table/users\n"
        + "return\n"
        + "rename node $user/UserId as QName('', 'ID'),\n"
        + "put(doc('../xml/users.xml'), 'new_users.xml')\n";
    xqExpr = xqconnection.prepareExpression(xquery);

    // execute the query
    xqExpr.executeQuery();
} finally {
    if (xqExpr != null) xqExpr.close();
    if (xqconnection != null) xqconnection.close();
}
}

```

To learn more about Stylus XQuery support for XQJ, see [Appendix G, “XQJ Support”](#).

Transforming Query Results

Transform expressions are used to create modified copies of nodes. Using `copy`, `modify`, and `return` clauses, transform expressions make a copy of an input document and then perform the XQuery and write the result to memory. Unlike other XUF update operations, transform expressions do not modify existing nodes.

Example – Replacing a Node Value

The `transform-change-values.xq` XQuery transforms the query result by increasing the number of `<Shares>` for user Minollo (`$holding/UserId eq 'Minollo'`) by 20% (replace value of node... with... (`$updatedHolding/Shares * 1.2`)). A new root element, `<table>`, is created for the modified nodes that result from this XQuery.

```
declare option ddtek:serialize "indent=yes";

<table> {
  for $holding in doc("../xml/holdings.xml")/table/holdings
  return
    if( $holding/UserId eq 'Minollo' ) then
      copy $updatedHolding := $holding
      modify
        replace value of node $updatedHolding/Shares with
          xs:integer($updatedHolding/Shares * 1.2)
      return $updatedHolding
    else
      $holding
} </table>
```

Example – Inserting a Node

The transform-insert-nodes.xq XQuery performs two update operations depending on the query result:

- If a user has DDTEK holdings, the value of the <Shares> node is increased by 1000
- If the user has no shares of DDTEK (else), a new <holdings> node is created for that user with the value of the <Shares> node set to 1000

```
declare option ddtek:serialize "indent=yes";
copy $newHoldings := doc("../xml/holdings.xml")
modify
  for $user in doc("../xml/users.xml")/table/users
  let $ddtekShares := $newHoldings/table/holdings[UserId eq $user/UserId and
    StockTicker eq "DDTEK"]
  return
    if( $ddtekShares ) then
      replace value of node $ddtekShares/Shares with $ddtekShares/Shares + 1000
    else
      insert node
        <holdings>
          <UserId>{$user/UserId/text()}</UserId>
          <StockTicker>DDTEK</StockTicker>
          <Shares>1000</Shares>
        </holdings>
      as last into $newHoldings/table
return $newHoldings
```

Updating Data Sources

Stylus XQuery provides a feature that allows XQuery to automatically update data sources that are accessed through `doc()` and `collection()` functions. You can use both literal and computed arguments for these function calls.

Relational data sources cannot be updated using automatic update.

Enabling Automatic Update

The *automatic update* feature can be enabled in one of two ways:

- Using the `ddtek:automatic-update` option. The automatic update feature can be enabled in any XQuery using the following declaration:

```
declare option ddtek:automatic-update "yes"
```
- Using the `-u` command line option. See [“Using the Command Line Utility” on page 40](#) for more information on using the command line.

The automatic update feature is disabled by default. When it is enabled, data sources affected by XUF update expressions are physically modified at the end of the XQuery execution.

In the event of a conflict between command line and XQuery settings for this feature, the setting specified at the XQuery level always takes precedence.

How Updates are Performed

Single data sources specified in a `doc()` function call are updated at the end of the XQuery execution. For a `collection()` function call that returns n XML documents, each of those n XML documents whose XDM instance has been modified is automatically updated.

Modified files are serialized using global serialization options set through the XQJ API or the `ddtek:serialize` option. See [Appendix D, “Serialization Support”](#) for more information.

Example

In the following example, the `<subject>` element in `books.xml` is renamed to `<topic>`. Changes to `books.xml` are saved automatically when the XQuery execution is complete:

```
declare option ddtek:automatic-update "yes";

for $book in doc("file:///c:/myFiles/my_books.xml")/books/book
return
  rename node $book/subject as QName("", "topic")
```


6 Understanding Data Sources and Connections

This chapter describes the XML and relational data sources that Stylus XQuery can work with to produce XML results, the methods available for connecting to these data sources, and how to configure URIs used to connect to them.

Using Data Sources in Queries

In XQuery, data sources and query results are both represented using an XML data model. Physical inputs such as XML text files, DOM trees, and relational databases are mapped into the XML data model when they are queried. In the case of relational databases, the mapping is logical, and relational data is not materialized as XML. These mappings are documented in this section.

The result of a query (the XML output) is also defined in the XML data model and must be mapped to a physical format such as DOM, SAX, StAX, or text in order for an application to use the results. See [“Accessing XML Results” on page 530](#) for more information about the mapping of results.

In addition, when you use Stylus XQuery with Stylus XML Converters, you can convert non-XML formats into XML, including EDI messages, tab-delimited and comma-separated text files, dBASE files, RTF files, and many more. Once these non-XML data sources are converted to XML, they are accessed the same as XML documents.

XML Data Sources

The XML data sources that Stylus XQuery can access have the following physical formats:

- XML text files/streams. These files/streams can be accessed using `fn:doc()`, which supports the `http:`, `ftp:`, and `file:` URI schemes and Stylus XML Converters URI schemes.

Here is an example of a file URI scheme:

```
let $request := doc('file:///c:/request.xml')/request
...
```

Here is an example of a Stylus XQuery XML Converters URI scheme in which the name of the Converter is Base64, the properties set for the conversion are newline and encoding, and the file to convert is `base_to_xml.bin`:

```
let $request := doc('converter:Base64:newline=crlf:
encoding=utf-8?file:///w:/myfiles/base_to_xml.bin')/
request
...
```

NOTE: Using `fn:collection()`, Stylus XQuery also allows you to make the following types of queries on XML files:

- Query multiple XML files in a directory. See [“Querying Multiple Files in a Directory” on page 288](#).
- Query XML files archived in ZIP or JAR files. See [“Querying ZIP, JAR, and MS Office Files” on page 292](#).

NOTES FOR SPECIFYING URIs:

- You must use forward slashes (/) in the path regardless of the platform.
- Relative URIs are allowed in the path. For example:

```
let $request := doc('request.xml')/request
...
```

- Spaces are allowed in the path. For example:

```
let $request := doc('file:///c:/DD XQuery/
request.xml')/request
...
```

If you use a custom URI resolver, the rules enforced for URI paths are governed by the syntax specified by the custom URI resolver (see [“Document URI Resolvers” on page 295](#)).

- XML contained in a Java application. This type of XML can be bound to the initial context item or to external variables in XQJ and used in XQuery queries. See [“Querying Data from XML Files or DOM Trees” on page 65](#) for an example.
- XML stored in columns of any supported relational database using an XML data type. See [Chapter 11 “Support for Relational Databases”](#) for details.
- XML stored in character columns of any supported relational database. See [“Querying XML Type Data” on page 263](#) for details.

Data Model Representation of XML Documents

For XQuery queries that access XML using `fn:doc()` or external variables in the query, Stylus XQuery implements the Infoset mapping described in the XQuery 1.0 and XPath 2.0 Data Model specification located at:

<http://www.w3.org/TR/2005/CR-xpath-datamodel-20051103/>

Relational Data Sources

Stylus XQuery provides support for the following relational databases:

- DB2
- DB2 Universal Database (UDB)
- Informix Dynamic Server
- Microsoft SQL Server
- MySQL Enterprise
- Oracle
- PostgreSQL
- Sybase Adaptive Server Enterprise

Note: For support for specific versions, see “[Supported Databases](#)” on page 445. This information is also maintained on:

https://www.xquery.com/data_sources

Specifying Relational Database Tables

Using Stylus XQuery, you specify relational database tables in a query using `fn:collection()`. The following example specifies a database table named `holdings`:

```
collection('holdings')
```

Notes for Specifying Database Table Names

- Qualified names – You might need to qualify the database table name in order to specify the exact table you want to access. The argument to `fn:collection()` can include any

combination of JDBC connection name, database catalog, or database schema in addition to the database table name.

For example, if the database table being accessed is not owned by the current user or is not located in the current database or catalog, you can qualify the database table name with the catalog name and schema name of the database table. The following example specifies a catalog name, schema name, and database table name:

```
collection('financial.joseph.holdings')
```

If you need to further qualify the database table name, you can use a *JDBC connection name*, which identifies a specific connection to a relational database. The following example specifies a JDBC connection name, catalog name, schema name, and database table name:

```
collection('stocks:financial.joseph.holdings')
```

- **Escape characters** – If the catalog name, schema name, or table name in the `fn:collection()` argument contains a period (`.`), a colon (`:`), or a backslash (`\`), escape the character with a backslash (`\`) so that Stylus XQuery can correctly parse the argument. For example, if the target table is named `a.holdings` and you specify the following query, Stylus XQuery parses `'a'` as the schema name, not as part of the table name:

```
collection('a.holdings')
```

Escaping the period (`.`) in the `fn:collection()` argument using the backslash character allows Stylus XQuery to parse the argument correctly:

```
collection('a\ holdings')
```

In addition, XQuery string literal syntax applies to the `fn:collection()` argument. If a table name contains double quotes, for example, `a"holdings`, and the `fn:collection()` argument uses double quote delimiters, you must repeat the double quotes:

```
collection("a""holdings")
```

Or, you can use:

```
collection('a"holdings')
```

See [“Path Expressions for Relational Sources” on page 81](#) for examples of using `fn:collection()`. See [“Resolving `fn:collection` Errors” on page 564](#) for information about troubleshooting `fn:collection()` errors.

Data Model Representation of Relational Tables

In Stylus XQuery, XML views of relational data are based on the SQL/XML mappings specified in the SQL 2003 standard. These are logical XML views, not a physical format.

SQL/XML provides different ways to parameterize views. Stylus XQuery uses the XMLFOREST variable to parameterize views. Using XMLFOREST is sufficient for most applications, but you can choose not to use it by setting the `JdbcSqlXmlForest` property of `DDXQDataSource` to false (see [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#)).

When the value of the `JdbcSqlXmlForest` property is set to true (the default), Stylus XQuery represents each database table by a sequence of row elements in a document node. The row elements use the table name as the element name and contain an element for each non-null column in the row; Stylus XQuery uses the column name as the element name for these elements. For example, when XML FOREST is used, the result for `collection('users')` is a document node containing the following elements:

```
<users>
  <userid>Minollo</userid>
  <firstname>Carlo</firstname>
  <lastname>Innocenti</lastname>
  <membersince>2004-06-16T00:00:00</membersince>
</users>
<users>
```



```

<userid>Jonathan</userid>
<firstname>Jonathan</firstname>
<lastname>Robie</lastname>
<othername>William</othername>
<membersince>2004-03-03T00:00:00</membersince>
</users>

```

Suppose we wanted to write a FLWOR expression to bind each row of the preceding table to a variable. In this case, we add an argument to `fn:collection()` that defines a path matching the `users` elements:

```
collection('users')/users
```

Here is a FLWOR expression that iterates over the rows returned by the preceding path expression:

```

for $u in collection('users')/users
where $u/lastname = 'Robie'
return $u/membersince

```

When the `XMLFOREST` variable is false, the result is a document node. Inside the document node is a single element that represents the table. Inside the single element node is a sequence of elements named `row`, each representing one row of the table. For example, when `XMLFOREST` is not used, a result for `collection('users')` might look like this:

```

<users>
  <row>
    <userid>Minollo</userid>
    <firstname>Carlo</firstname>
    <lastname>Innocenti</lastname>
    <membersince>2004-06-16T00:00:00</membersince>
  </row>
  <row>
    <userid>Jonathan</userid>
    <firstname>Jonathan</firstname>
    <lastname>Robie</lastname>
    <othername>William</othername>
    <membersince>2004-03-03T00:00:00</membersince>
  </row>
</users>

```

```
</row>
</users>
```

Suppose we wanted to write a FLWOR expression to bind each row of the preceding table to a variable. In this case, we would need to add a path expression to `fn:collection()`. The following path expression defines a path that matches the row elements:

```
collection('users')/users/row
```

Here is a FLWOR expression that iterates over the rows returned by the preceding path expression:

```
for $u in collection('users')/users/row
where $u/lastname = 'Robie'
return $u/membersince
```

Case Sensitivity

XML element and attribute names are case-sensitive. When SQL column and table names are mapped into XML elements, the case depends on a number of factors that can vary by database vendor and the parameters used to create a database.

The case sensitivity of the argument to `fn:collection()` depends on the database.

Data Type Mappings

See [“Data Type Mappings” on page 447](#) for information about how database data types are mapped to XML schema data types.

Security Features

Stylus XQuery supports authentication and data encryption security features for data source connections. For more information, see [Chapter 7, “Securing Data Source Connections.”](#)

Choosing a Connection Method

You can use XQJ to connect to a data source using either of the following methods:

- Construct a `DDXQDataSource` instance in your Java application explicitly
- Load a `DDXQDataSource` object from JNDI

Specifying connection information explicitly in your Java application using a `DDXQDataSource` instance requires you to code the connection information directly in your Java application. See [“Configuring Connections Explicitly” on page 123](#) for complete information.

Using a `DDXQDataSource` object loaded from JNDI can be a convenient way to manage connections because the connection information is created and managed outside the applications that use it. As a result, the effort required to reconfigure your environment when an infrastructure change occurs is minimal. For example, if a database is moved to another server and uses a different port number, you only need to change the relevant properties of the data source object. An application accessing the database does not need to change because the application only references the logical name of the data source object in JNDI. See [“Configuring Connections Using JNDI” on page 127](#) for complete information.

Configuring Connections Explicitly

To specify connection information explicitly using XQJ, construct an `XQDataSource` instance in your Java application using the `DDXQDataSource` class.

If your Java application contains queries that access an XML file, you can directly access the file as shown in the following XQJ code, where the location and name of the XML file is specified as a parameter of `fn:doc()`, an XQuery function.

```
DDXQDataSource ds = new DDXQDataSource();
XQConnection conn = ds.getConnection();
conn.createExpression().executeQuery("doc('path_and_filename')");
```

How you configure connection information for relational databases using XQJ depends on whether you are accessing a single database or multiple databases. If your Java application contains XQuery queries that access a single database, you can configure connection information using the `DDXQDataSource` class as shown in the following XQJ code:

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

If your Java application contains XQuery queries that access multiple databases, use the `DDXQJDBCConnection` class to configure connection information for each database connection, then construct an `XQDataSource` instance that specifies all database connections using the `DDXQDataSource` class as shown in the following XQJ code. When specifying the URI for multiple databases, use the `DDXQJDBCConnection` `Url` property instead of the `DDXQDataSource` `jdbcUrl` property to set the JDBC URI for each connection.

```
DDXQJDBCConnection jc1 = new DDXQJDBCConnection();
jc1.setUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
DDXQJDBCConnection jc2 = new DDXQJDBCConnection();
jc2.setUrl("jdbc:xquery:sqlserver://server2:1433;databaseName=holdings");
DDXQDataSource ds = new DDXQDataSource();
ds.setDdxqJdbcConnection(new DDXQJDBCConnection[] {jc1, jc2});
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

In the preceding example, notice that the user name and password specified in the `getConnection()` method are used to establish all underlying JDBC connections. If you require different

user names and passwords for each connection, set the user name and password on each DDXQJDBCCConnection as shown in the following XQJ code:

```
DDXQJDBCCConnection jc1 = new DDXQJDBCCConnection();
jc1.setUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
jc1.setUser("myuserid1");
jc1.setPassword("mypswd1");
DDXQJDBCCConnection jc2 = new DDXQJDBCCConnection();
jc2.setUser("myuserid2");
jc2.setPassword("mypswd2");
jc2.setUrl("jdbc:xquery:sqlserver://server2:1433;databaseName=holdings");
DDXQDataSource ds = new DDXQDataSource();
ds.setDdxqJdbcConnection(new DDXQJDBCCConnection[] {jc1, jc2});
XQConnection conn = ds.getConnection();
```

The following table lists properties of the DDXQDataSource class and the corresponding properties of the DDXQJDBCCConnection class:

DDXQDataSource property	DDXQJDBCCConnection property
JdbcName	Name
JdbcOptions	Options
JdbcSqlXmlForest	SqlXmlForest
JdbcSqlXmlIdentifierEscaping	SqlXmlIdentifierEscaping
JdbcTempTableColumns	TempTableColumns
JdbcTempTableSuffix	TempTableSuffix
JdbcTransactionIsolationLevel	TransactionIsolationLevel
JdbcUrl	Url
Password	Password
User	User

If any of these DDXQJDBCCConnection properties is specified for an individual connection and then specified again using DDXQDataSource, the latter overrides the former. The following

example shows the `Options` property first specified for the `jc1` connection, and specified again using the `JdbcOptions` property of `DDXQDataSource`. In this case, the precision and scale for `xs:decimal` specified for `DDXQDataSource` overrides the precision and scale for `xs:decimal` specified for the `jc1` connection.

```
DDXQJDBCConnection jc1 = new DDXQJDBCConnection();
jc1.setUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
jc1.setUser("myuserid1");
jc1.setPassword("mypswd1");
jc1.setOptions("sql-decimal-cast=25,5");
DDXQJDBCConnection jc2 = new DDXQJDBCConnection();
jc2.setUser("myuserid2");
jc2.setPassword("mypswd2");
jc2.setUrl("jdbc:xquery:sqlserver://server2:1433;databaseName=holdings");
DDXQDataSource ds = new DDXQDataSource();
ds.setDdxqJdbcConnection(new DDXQJDBCConnection[] {jc1, jc2});
ds.setJdbcOptions("sql-decimal-cast=35,20");
XQConnection conn = ds.getConnection();
```

See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#) for a description of properties you can set using XQJ.

Configuring Connections Using JNDI

To create your own data source object for JNDI to use with Stylus XQuery, you can use the example named `JNDIDataSource` in the `/examples` subdirectory in your Stylus XQuery installation directory as a template.

Once you have created a data source object, you can register it with JNDI, as shown in the following XQJ code, where `holdings_ds` is the name of the data source object:

```
DDXQDataSource ds = new DDXQDataSource();  
Context ctx = new InitialContext();  
ctx.bind("holdings_ds", ds);
```

The following XQJ code shows how to load a `DDXQDataSource` object from JNDI:

```
Context ctx = new InitialContext();  
XQDataSource ds = (XQDataSource)ctx.lookup("holdings_ds");  
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the name of the data source object (`holdings_ds`). The `Context.lookup()` method returns a reference to a Java object, which is cast to a `com.ddtek.xquery.xqj.DDXQDataSource` object. Finally, the `getConnection()` method is called to establish the connection.

DDXQDataSource and DDXQJDBCCConnection Properties

The class name of the Stylus XQuery XQDataSource implementation is:

```
com.ddtek.xquery.xqj.DDXQDataSource
```

It provides properties that allow you to configure most Stylus XQuery settings and data source connections. [Table 6-1](#) lists the properties supported by the DDXQDataSource class, Stylus XQuery’s XQDataSource implementation, and describes each property.

The following class provides additional properties for configuring connections to *multiple* databases:

```
com.ddtek.xquery.xqj.DDXQJDBCCConnection
```

See [Table 6-2](#) for a list of the additional properties supported by the DDXQJDBCCConnection class.

NOTE: All property names are case-insensitive. For example, password is the same as Password.

DDXQDataSource Properties

Table 6-1. DDXQDataSource Properties

Property	Description
AllowJavaFunctions	Specifies whether external Java functions are allowed. By default, external functions are allowed. To disable external functions for security reasons, for example, set this property to false.

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
BaseUri	The baseUri property in the XQuery static context, which is the base URI used to resolve relative URIs in fn:doc(). See “XML Data Sources” on page 116 for rules governing URIs. NOTE: You can also specify a base URI in the query prolog.
Collation	The collation URI for the default collation to be used by the Java Virtual Machine. See “Querying Multiple Files in a Directory” on page 288 for more information.
CollectionUriResolver	A Java class that implements the com.ddtek.xquery.CollectionUriResolver interface to resolve URIs in fn:collection(). For example, you may want to create a Java class to resolve custom URLs that point to a directory that contains your XML files. See “Collection URI Resolvers” on page 298 for more information.
DocumentUriResolver	A Java class that implements the javax.xml.transform.URIResolver interface to resolve URIs in fn:doc() and fn:doc-available(). For example, you may want to create a Java class to resolve custom URLs that point to a proprietary repository that stores your XML documents such as an XML database. See “Document URI Resolvers” on page 295 for more information.
JdbcName	The name of the JDBC connection. A JDBC connection name identifies a specific connection to a relational database. If specifying a JDBC connection name for multiple databases, use the Name property of the DDXQJDBCConnection class.

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
JdbcOptions	<p>Specifies one or multiple option declarations for the relational database specified by the JdbcUrl property. Valid option declarations are:</p> <ul style="list-style-type: none">sql-decimal-castsql-extra-checks-trailing-spacessql-ignore-trailing-spacessql-ora10-use-binary-float-doublesql-order-by-on-valuessql-rewrite-algorithmsql-rewrite-exists-into-countsql-simple-convert-functionssql-simple-string-functionssql-sybase-temptable-indexsql-sybase-use-derived-tablessql-unicode-stringssql-varchar-cast <p>See “Option Declarations” on page 275 for a description of these option declarations.</p> <p>The value of this property is a semicolon-separated list of option declaration name=value pairs:</p> <pre>name=value[;name=value]...</pre> <p>For example:</p> <pre>sql-unicode-strings=yes;sql-decimal-cast=10,5</pre> <p>You also can specify a global option declaration for all XML and relational data sources using the Options property.</p> <p>NOTE: You can override this setting in the query.</p> <p>If specifying an option declaration for multiple databases, use the Options property of the DDXQJDBCConnection class.</p>
JdbcSqlXmlForest	<p>Specifies the format of the XML result that fn:collection() returns. Valid values are true and false. The default is true. See “Data Model Representation of Relational Tables” on page 120 for details about the format of the XML result.</p>

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
JdbcSqlXmlIdentifierEscaping	<p>Specifies how Stylus XQuery handles escaping of identifiers, which is needed because of mismatches that occur when characters in SQL identifiers are mapped to XML. Valid values are:</p> <ul style="list-style-type: none"> ■ none (the default) – No mapping is performed. An error is raised if a character in a SQL identifier cannot be mapped to XML. ■ partial – Characters in SQL identifiers that are not XML characters are escaped using an underscore character (_) followed by a lowercase x followed by the character's Unicode representation in hexadecimal format followed by an underscore character (_). For example, sales!forecast becomes sales_x0021_forecast. ■ full – In addition to the escaping performed by partial, the character x of a SQL identifier that starts with "xml" (in any combination of upper and lowercase characters) is escaped. For example, XMLTable becomes _x0058_MLTable.

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
JdbcTempTableColumns	<p><code>xqueryType="value" sqlType="value"</code>. Specifies which SQL type is used for columns when Stylus XQuery creates temporary tables for query optimization.</p> <p>If you do not specify a value for this property, Stylus XQuery uses the SQL/XML mappings to determine which SQL type to use for the columns of temporary tables; however, this can sometimes cause problems. For example, if your database table has a case-sensitive collation and the temporary table is created with a case-insensitive collation, an error is raised. In this case, use this property to specify that the temporary tables be created with a case-sensitive collation.</p> <p>A value for <code>xqueryType</code> is required and specifies one of the following values: <code>boolean</code>, <code>byte</code>, <code>date</code>, <code>dateTime</code>, <code>decimal</code>, <code>double</code>, <code>float</code>, <code>hexBinary</code>, <code>int</code>, <code>integer</code>, <code>long</code>, <code>short</code>, <code>string</code>, or <code>time</code>.</p> <p>A value for <code>sqlType</code> is required and determines the database type declaration that is appended to the column names when temporary tables are created. The specified data type must be supported by the database used to create the temporary tables.</p> <p>For example:</p> <pre>xqueryType="string" sqlType="nvarchar(2000) collate SQL_Latin1_General_CP1_CS_AS"</pre>
JdbcTempTableSuffix	<p><code>CCSID UNICODE ON COMMIT PRESERVE ROWS</code>. You must specify this property if you are connecting to a DB2 for z/OS Unicode database.</p>

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
JdbcTransactionIsolationLevel	<p>Specifies the transaction isolation level. Valid values are:</p> <ul style="list-style-type: none"> ■ <code>java.sql.Connection.TRANSACTION_READ_UNCOMMITTED</code> – Locks are obtained on modifications to the database and held until end of transaction (EOT). Reading from the database does not involve any locking. ■ <code>java.sql.Connection.TRANSACTION_READ_COMMITTED</code> – Locks are acquired for reading and modifying the database. Locks are released after reading, but locks on modified objects are held until EOT. ■ <code>java.sql.Connection.TRANSACTION_REPEATABLE_READ</code> – Locks are obtained for reading and modifying the database. Locks on all modified objects are held until EOT. Locks obtained for reading data are held until EOT. Locks on non-modified access structures (such as indexes and hashing structures) are released after reading. ■ <code>java.sql.Connection.TRANSACTION_SERIALIZABLE</code> – All data read or modified is locked until EOT. All access structures that are modified are locked until EOT. Access structures used by the query are locked until EOT. ■ <code>java.sql.Connection.TRANSACTION_NONE</code> – Transactions are not supported. ■ <code>-1</code> – The default transaction isolation level is used, which is Read Committed. <p>The database to which you are connecting may not support all of these isolation levels. See “Transaction Isolation Levels” on page 271 for details.</p> <p>NOTE: Once a connection is made, the transaction isolation level cannot be changed for that connection (XQConnection object).</p>

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
JdbcUrl	<p>The JDBC URL of the database. See “Specifying Connection URIs” on page 141 for the syntax of URLs.</p> <p>If specifying a URL for multiple databases, use the <code>Url</code> property of the <code>DDXQJDBCCConnection</code> class.</p>
MaxPooledQueries	<p>Specifies the maximum number of queries that can be placed in the pool when Stylus XQuery’s internal query pooling is enabled. When enabled, Stylus XQuery caches a certain number of queries executed by an application. For example, if this property is set to 20, Stylus XQuery caches the last 20 queries executed by the application. If the value set for this property is greater than the number of queries used by the application, all queries are cached. By default, query pooling is disabled. See “Using Query Pooling” on page 192 for more information.</p>
ModuleUriResolver	<p>A Java class that implements the <code>com.ddtek.xquery.ModuleURIResolver</code> interface to resolve the library module to be imported. For example, you may want to create a Java class to resolve URLs that point to a custom repository that stores XQuery modules. See also “Library Module URI Resolvers” on page 296.</p>
Options	<p>Specifies a global option declaration to use as the default for all XML and relational data sources that are used by XQuery queries in your Java application. Valid global option declarations are:</p> <ul style="list-style-type: none">■ <code>detect-XPST0005</code>■ <code>plan-explain</code>■ <code>serialize</code>■ <code>xml-streaming</code> <p>See “Option Declarations” on page 275 for a description of these option declarations.</p> <p>The value of this property is a <code>name=value</code> pair:</p> <p><i>name=value</i></p> <p>where <i>value</i> is either <code>yes</code> or <code>no</code>. For example:</p> <p><code>detect-XPST0005=no</code></p>

Table 6-1. DDXQDataSource Properties (cont.)

Property	Description
Password	<p>A password used to connect to the database.</p> <p>If specifying a password for multiple databases, use the Password property of the DDXQJDBCConnection class.</p>
SpyAttributes	<p>Enables and sets attributes for Stylus Spy, a tool that logs detailed information about XQJ calls issued by a running Java application. For example, you may want to log all XQJ activity to a log file on your local machine.</p> <p>The format for the value of this property is:</p> <pre>(spy_attribute=value[;spy_attribute=value]...)</pre> <p>where <i>spy_attribute=value</i> is a Stylus Spy attribute and a valid value for that attribute. The following example specifies that Stylus Spy log all XQJ activity to a log file, including the content of SAX streams passed through XQJ.</p> <pre>ds.setSpyAttributes("log=(file)/tmp/spy.log; logSAX=yes")</pre> <p>NOTE: When coding a path in a Java string on Windows, the backslash character (\) must be preceded by the Java escape character, which is also a backslash. The Spy parser also uses the backslash as an escape character, so four slashes must be used to specify a single backslash in the log path. For example:</p> <pre>ds.setSpyAttributes("log= (file)C:\\\\temp\\\\spy.log;logSAX=yes")</pre> <p>Once enabled, you can turn Stylus Spy on and off at runtime using the <code>setEnabledLogging()</code> method of the <code>com.ddtek.xquery.xqj.ExtLogControl</code> interface.</p> <p>See “Logging XQJ Calls with Stylus Spy™ for XQJ” on page 553 for instructions on using Stylus Spy and a list of supported attributes.</p>
User	<p>A user name used to connect to the database.</p> <p>If specifying a user for multiple databases, use the User property of the DDXQJDBCConnection class.</p>

DDXQJDBCCConnection Properties

Table 6-2 lists the properties supported by the DDXQJDBCCConnection class and describes each property (see Table 6-1 for a list of the properties supported by the DDXQDataSource class).

Table 6-2. DDXQJDBCCConnection Properties

Property	Description
Name	The name of the JDBC connection. A JDBC connection name identifies a specific connection to a relational database. If specifying a JDBC connection name for a single database, use the JdbcName property of the DDXQDataSource class.

Table 6-2. DDXQJDBCConnection Properties (cont.)

Property	Description
Options	<p>Specifies one or multiple option declarations for the relational database specified by the Url property. Valid option declarations are:</p> <ul style="list-style-type: none"> ■ sql-decimal-cast ■ sql-extra-checks-trailing-spaces ■ sql-ignore-trailing-spaces ■ sql-ora10-use-binary-float-double ■ sql-order-by-on-values ■ sql-rewrite-algorithm ■ sql-rewrite-exists-into-count ■ sql-simple-convert-functions ■ sql-simple-string-functions ■ sql-sybase-temptable-index ■ sql-sybase-use-derived-tables ■ sql-unicode-strings ■ sql-varchar-cast <p>See “Option Declarations” on page 275 for a description of these option declarations.</p> <p>The value of this property is a semicolon-separated list of option declaration name=value pairs:</p> <pre>name=value[;name=value]...</pre> <p>For example:</p> <pre>sql-unicode-literals=yes;sql-decimal-cast=10,5</pre> <p>You also can specify a global option declaration for all XML and relational data sources using the Options property of the DDXQDataSource class.</p> <p>NOTE: You can override this setting in the query.</p> <p>If specifying an option declaration for a single database, use the JdbcOptions property of the DDXQDataSource class.</p>
Password	<p>A password used to connect to the database. A password is required only if security is enabled on the database. Contact your system administrator to obtain your password.</p> <p>If specifying a password for a single database, use the Password property of the DDXQDataSource class.</p>

Table 6-2. DDXQJDBCCONNECTION Properties (cont.)

Property	Description
SqlXmlForest	Specifies the format of the XML result that fn:collection() returns. Valid values are true and false. The default is true. See “Data Model Representation of Relational Tables” on page 120 for details about the format of the XML result.
SqlXmlIdentifierEscaping	<p>Specifies how Stylus XQuery handles escaping of identifiers, which is needed because of mismatches that occur when characters in SQL identifiers are mapped to XML. Valid values are:</p> <ul style="list-style-type: none">■ none (the default) – No mapping is performed. An error is raised if a character in a SQL identifier cannot be mapped to XML.■ partial – Characters in SQL identifiers that are not XML characters are escaped using an underscore character (_) followed by a lowercase x followed by the character’s Unicode representation in hexadecimal format followed by an underscore character (_). For example, sales!forecast becomes sales_x0021_forecast.■ full – In addition to the escaping performed by partial, the character x of a SQL identifier that starts with "xml" (in any combination of upper and lowercase characters) is escaped. For example, XMLTable becomes _x0058_MLTable.

Table 6-2. DDXQJDBCConnection Properties (cont.)

Property	Description
TempTableColumns	<p><code>xquerytype="value" sqlType="value"</code>. Specifies which SQL type is used for columns when Stylus XQuery creates temporary tables for query optimization.</p> <p>If you do not specify a value for this property, Stylus XQuery uses the SQL/XML mappings to determine which SQL type to use for the columns of temporary tables; however, this can sometimes cause problems. For example, if your database table has a case-sensitive collation and the temporary table is created with a case-insensitive collation, an error is raised. In this case, use this property to specify that the temporary tables be created with a case-sensitive collation.</p> <p>A value for <code>xqueryType</code> is required and specifies one of the following values: <code>boolean</code>, <code>byte</code>, <code>date</code>, <code>dateTime</code>, <code>decimal</code>, <code>double</code>, <code>float</code>, <code>hexBinary</code>, <code>int</code>, <code>integer</code>, <code>long</code>, <code>short</code>, <code>string</code>, or <code>time</code>.</p> <p>A value for <code>sqlType</code> is required and determines the database type declaration that is appended to the column names when temporary tables are created. The specified data type must be supported by the database used to create the temporary tables.</p> <p>For example:</p> <pre>xqueryType="string" sqlType="nvarchar(2000) collate SQL_Latin1_General_CP1_CS_AS"</pre>
TempTableSuffix	<p><code>CCSID UNICODE ON COMMIT PRESERVE ROWS</code>. You must specify this property if you are connecting to a DB2 for z/OS Unicode database.</p>

Table 6-2. DDXQJDBCCConnection Properties (cont.)

Property	Description
TransactionIsolationLevel	<p>Specifies the transaction isolation level. Valid values are:</p> <ul style="list-style-type: none">■ java.sql.Connection.TRANSACTION_READ_UNCOMMITTED – Locks are obtained on modifications to the database and held until end of transaction (EOT). Reading from the database does not involve any locking.■ java.sql.Connection.TRANSACTION_READ_COMMITTED – Locks are acquired for reading and modifying the database. Locks are released after reading, but locks on modified objects are held until EOT.■ java.sql.Connection.TRANSACTION_REPEATABLE_READ – Locks are obtained for reading and modifying the database. Locks on all modified objects are held until EOT. Locks obtained for reading data are held until EOT. Locks on non-modified access structures (such as indexes and hashing structures) are released after reading.■ java.sql.Connection.TRANSACTION_SERIALIZABLE – All data read or modified is locked until EOT. All access structures that are modified are locked until EOT. Access structures used by the query are locked until EOT.■ java.sql.Connection.TRANSACTION_NONE – Transactions are not supported.■ -1 – The default transaction isolation level is used, which is Read Committed. <p>The database to which you are connecting may not support all of these isolation levels. See “Transaction Isolation Levels” on page 271 for details.</p> <p>NOTE: Once a connection is made, the transaction isolation level cannot be changed for that connection (XQConnection object).</p>

Table 6-2. DDXQJDBCConnection Properties (cont.)

Property	Description
Url	<p>The JDBC URL of the database. See “Specifying Connection URIs” on page 141 for the syntax of URLs.</p> <p>If specifying a URL for a single database, use the <code>JdbcUrl</code> property of the <code>DDXQDataSource</code> class.</p>
User	<p>A user name used to connect to the database.</p> <p>If specifying a user name for a single database, use the <code>User</code> property of the <code>DDXQDataSource</code> class.</p>

Specifying Connection URIs

Stylus XQuery provides access to most databases through built-in JDBC drivers. In addition, you can access specific databases using third-party JDBC drivers. The format of the connection URI depends on whether you are using a built-in JDBC driver or a third-party driver, and the database you are connecting to.

Connection URIs for Built-In Drivers

Stylus XQuery provides built-in JDBC drivers to access the following databases:

- DB2 for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 for iSeries
- Informix Dynamic Server
- MySQL Enterprise
- Oracle
- Microsoft SQL Server
- Sybase Adaptive Server Enterprise

The format of the connection URI is:

```
jdbc:xquery:dbtype://server_name:port;property=value[;...]
```

where:

<i>dbtype</i>	Valid values are db2, informix, mysql, oracle, sqlserver, and sybase.
<i>server_name</i>	The TCP/IP address or TCP/IP host name of the database server to which you are connecting (See following NOTE).
<i>port</i>	The number of the TCP/IP port.
<i>property=value</i>	Connection properties. For a list of connection properties, ordered by database, see “Database Connection Properties” on page 460 . For some databases, particular connection properties are required in the URL as shown in the following examples. All connection property names are case-insensitive. For example, password is the same as Password.

NOTE FOR ORACLE USERS: See [“Using Oracle tnsnames.ora Files” on page 476](#) for instructions on retrieving connection information from an Oracle tnsnames.ora file.

The following URLs show examples of the minimum information, including any required connection properties, that must be specified in a connection URL.

DB2 for Linux/UNIX/Windows

```
jdbc:xquery:db2://server_name:50000;databaseName=your_database
```

DB2 for z/OS and iSeries

```
jdbc:xquery:db2://server_name:446;locationName=db2_location
```

Informix

```
jdbc:xquery:informix://server_name;1526;InformixServer=dbserver_name
```

Microsoft SQL Server

```
jdbc:xquery:sqlserver://server_name:1433
```

MySQL Enterprise

```
jdbc:xquery:mysql://server_name:[port]
```

Oracle

```
jdbc:xquery:oracle://server_name:1521
```

Sybase

```
jdbc:xquery:sybase://server_name:5000
```

Connection URIs for Third-Party Drivers

You can access the PostgreSQL database using the PostgreSQL JDBC driver from:

<http://jdbc.postgresql.org/>

Connection URI Format

The format of the connection URI is:

```
jdbc:postgresql:database
```

or

```
jdbc:postgresql://[server_name][:port]/database[?property=value]
[&property=value[...]]
```

where:

<i>server_name</i>	<p>The TCP/IP address or TCP/IP host name of the database server to which you are connecting. If an IPv6 TCP/IP address is specified, it must be enclosed within brackets. For example:</p> <pre>jdbc:postgresql://[::1]:5432/accounting</pre> <p>If an address or a host name is not specified, the value defaults to a host name of <code>localhost</code>.</p>
<i>port</i>	<p>The number of the TCP/IP port. If a port is not specified, the value defaults to 5432.</p>
<i>database</i>	<p>The name of the database you are connecting to.</p>
<i>property=value</i>	<p>Connection properties. Refer to your PostgreSQL JDBC driver documentation for information about the connection properties supported by the driver.</p>

The following URI is an example of the minimum information that must be specified in the URI:

```
jdbc:postgresql:your_database
```


7 Securing Data Source Connections

Stylus XQuery supports these security methods:

- Authentication
- Data encryption

This chapter describes these methods and how to implement them. It covers the following topics:

- [About Authentication](#)
- [Using Kerberos Authentication](#)
- [Using NTLM Authentication](#)
- [Data Encryption Across the Network](#)

About Authentication

On most computer systems, a password is used to prove (authenticate) a user's identity. This password often is transmitted over the network and can possibly be intercepted by malicious hackers. Because this password is the one secret piece

of information that identifies a user, anyone knowing a user's password can effectively *be* that user.

Authentication methods protect the identity of the user. Stylus XQuery supports the following authentication methods:

- User ID/password authentication authenticates the user to the database using a database user name and password.
- Kerberos is a trusted third-party authentication service. The drivers support both Windows Active Directory Kerberos and MIT Kerberos implementations for DB2, Oracle, and Sybase. For Microsoft SQL Server, the driver supports Windows Active Directory Kerberos only.
- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The database server relies on the client to authenticate the user and does not provide additional authentication.
- NTLM authentication is a single sign-on authentication method for Windows environments. This method provides authentication from Windows clients only.

[Table 7-1](#) shows the authentication methods supported by Stylus XQuery.

Table 7-1. Authentication Methods Supported by Stylus XQuery

Driver	User ID/ Password	Kerberos ^a	Client	NTLM
DB2 for Linux/UNIX/Windows	X	X	X	
DB2 for z/OS	X	X	X	
DB2 for iSeries	X		X	
Informix	X			
MySQL	X			
Oracle	X	X	X	X

Table 7-1. Authentication Methods Supported by Stylus XQuery

Driver	User ID/ Password	Kerberos ^a	Client	NTLM
Microsoft SQL Server	X	X ^b		X
Sybase	X	X		

- a. For DB2, Oracle, and Sybase, the drivers support the Windows Active Directory KDC and MIT Kerberos KDC. For Microsoft SQL Server, the driver supports the Windows Active Directory KDC only.
- b. Supported for Microsoft SQL Server 2000 and higher.

Using Kerberos Authentication

Kerberos authentication is a trusted third-party authentication service. Kerberos authentication can take advantage of the user name and password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application.

Stylus XQuery supports Kerberos authentication for the following databases:

- DB2
- Oracle
- Microsoft SQL Server
- Sybase

Verify that your environment meets the requirements listed in [Table 7-2](#) before you configure Stylus XQuery for Kerberos authentication.

Table 7-2. Kerberos Authentication Requirements

Component	Requirements
Database server	<p>The database server must be running one of the following databases:</p> <p>DB2:</p> <ul style="list-style-type: none">■ DB2 v8.1 or higher for Linux/UNIX/Windows <p>Oracle:</p> <ul style="list-style-type: none">■ Oracle 11g■ Oracle 10g (R1 and R2) <p>Microsoft SQL Server:</p> <ul style="list-style-type: none">■ Microsoft SQL Server 2008■ Microsoft SQL Server 2005■ Microsoft SQL Server 2000■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher <p>Sybase:</p> <ul style="list-style-type: none">■ Sybase 12.5.1 or higher

Table 7-2. Kerberos Authentication Requirements*(cont.)*

Component	Requirements
Kerberos server	<p>The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos Key Distribution Center (KDC). If using Windows Active Directory, this machine is also the domain controller.</p> <p>DB2, Oracle, and Sybase:</p> <p>Network authentication must be provided by one of the following methods:</p> <ul style="list-style-type: none">■ Windows Active Directory on one of the following operating systems:<ul style="list-style-type: none">• Windows Server 2003• Windows 2000 Server Service Pack 3 or higher■ MIT Kerberos 1.4.2 or higher <p>Microsoft SQL Server:</p> <p>Network authentication must be provided by Windows Active Directory on one of the following operating systems:</p> <ul style="list-style-type: none">■ Windows Server 2003■ Windows 2000 Server Service Pack 3 or higher

Configuring Kerberos Authentication

During installation, Stylus XQuery installs the following files required for Kerberos authentication in the /lib subdirectory of your Stylus XQuery installation directory:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. Stylus XQuery installs a generic file that you must modify for your environment.
- JDBCDriverLogin.conf file is a configuration file that specifies which Java Authentication and Authorization Service (JAAS) login module to use for Kerberos authentication. This file is

configured to load automatically unless the `java.security.auth.login.config` system property is set to load another configuration file. You can modify this file, but Stylus XQuery must be able to find the `JDBC_DRIVER_01` entry in this file or another specified login configuration file to configure the JAAS login module. Refer to your J2SE documentation for information about setting configuration options in this file

To configure Stylus XQuery:

- 1 Set the `AuthenticationMethod` connection property to `kerberos`. See the DB2, Oracle, Microsoft SQL Server, and Sybase connection properties tables in [“Database Connection Properties” on page 460](#) for more information about setting a value for this property.
- 2 Modify the `krb5.conf` file to contain your Kerberos realm name and the KDC name for that Kerberos realm by editing the file with a text editor or by specifying the system properties, `java.security.krb5.realm` and `java.security.krb5.kdc`.

NOTE: If using Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.

For example, if your Kerberos realm name is `XYZ.COM` and your KDC name is `kdc1`, your `krb5.conf` file would look like this:

```
[libdefaults]
    default_realm = XYZ.COM

[realms]
    XYZ.COM = {
        kdc = kdc1
    }
```

If the `krb5.conf` file does not contain a valid Kerberos realm and KDC name, an exception is thrown.

- 3 If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and Stylus XQuery. See [“Permissions for Kerberos Authentication” on page 151](#) for an example.

Permissions for Kerberos Authentication

Using Stylus XQuery on a Java 2 Platform with the standard Security Manager enabled requires certain permissions to be set in the security policy file of the Java 2 Platform. This security policy file can be found in the `jre/lib/security` subdirectory of the Java 2 Platform installation directory.

NOTE: Web browser applets running in the Java 2 plug-in are always running in a JVM with the standard Security Manager enabled.

To run an application on a Java 2 Platform with the standard Security Manager, use the following command:

```
"java -Djava.security.manager application_class_name"
```

where *application_class_name* is the class name of the application.

Refer to your Java 2 Platform documentation for more information about setting permissions in the security policy file.

To use Kerberos authentication with Stylus XQuery, the application and code bases must be granted security permissions in the security policy file of the Java 2 Platform as shown in the following examples.

DB2

```
grant codeBase "file:/install_dir/lib/-" {
    permission java.lang.RuntimePermission "getProtectionDomain";
    permission java.util.PropertyPermission "java.security.krb5.conf", "read";
    permission java.util.PropertyPermission "java.security.auth.login.config",
        "read", "write";
    permission javax.security.auth.AuthPermission
        "createLoginContext.JDBC_DRIVER_01";
    permission javax.security.auth.AuthPermission
        "createLoginContext.DDTEK-JDBC";
    permission javax.security.auth.AuthPermission "doAs";
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/your_realm@your_realm", "initiate";
    permission javax.security.auth.kerberos.ServicePermission
        "principal_name/db_hostname@your_realm", "initiate";
};
```

where:

install_dir is the Stylus XQuery installation directory.

principal_name is the service principal name registered with the Kerberos Key Distribution Center (KDC) that identifies the database service.

your_realm is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

db_hostname is the host name of the machine running the database.

Oracle

```
grant codeBase "file:/install_dir/lib/-" {
  permission java.lang.RuntimePermission "getProtectionDomain";
  permission java.util.PropertyPermission "java.security.krb5.conf", "read";
  permission java.util.PropertyPermission "java.security.auth.login.config",
    "read", "write";
  permission javax.security.auth.AuthPermission
    "createLoginContext.JDBC_DRIVER_01";
  permission javax.security.auth.AuthPermission
    "createLoginContext.DDTEK-JDBC";
  permission javax.security.auth.AuthPermission "doAs";
  permission javax.security.auth.kerberos.ServicePermission
    "krbtgt/your_realm@your_realm", "initiate";
  permission javax.security.auth.kerberos.ServicePermission
    "principal_name/db_hostname@your_realm", "initiate";
};
```

where:

install_dir is the Stylus XQuery installation directory.

your_realm is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

principal_name is the service principal name registered with the Kerberos Key Distribution Center (KDC) that identifies the database service.

db_hostname is the host name of the machine running the database.

Microsoft SQL Server

```
grant codeBase "file:/install_dir/lib/-" {
    permission java.lang.RuntimePermission "getProtectionDomain";
    permission java.util.PropertyPermission "java.security.krb5.conf", "read";
    permission java.util.PropertyPermission "java.security.auth.login.config",
        "read", "write";
    permission javax.security.auth.AuthPermission
        "createLoginContext.JDBC_DRIVER_01";
    permission javax.security.auth.AuthPermission
        "createLoginContext.DDTEK-JDBC";
    permission javax.security.auth.AuthPermission "doAs";
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/your_realm@your_realm", "initiate";
    permission javax.security.auth.kerberos.ServicePermission
        "MSSQLSvc/db_hostname:SQLServer_port@your_realm", "initiate";
};
```

where:

install_dir is the Stylus XQuery installation directory.

your_realm is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

db_hostname is the host name of the machine running the database.

SQLServer_port is the TCP/IP port on which the Microsoft SQL Server instance is listening.

Sybase

```
grant codeBase "file:/install_dir/lib/-" {
permission java.lang.RuntimePermission "getProtectionDomain";
    permission java.util.PropertyPermission "java.security.krb5.conf", "read";
    permission java.util.PropertyPermission "java.security.auth.login.config",
        "read", "write";
    permission javax.security.auth.AuthPermission
        "createLoginContext.JDBC_DRIVER_01";
    permission javax.security.auth.AuthPermission
        "createLoginContext.DDTEK-JDBC";
    permission javax.security.auth.AuthPermission "doAs";
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/your_realm@your_realm", "initiate";
    permission javax.security.auth.kerberos.ServicePermission
        "principal_name/db_hostname@your_realm", "initiate";
};
```

where:

install_dir is the Stylus XQuery installation directory.

your_realm is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

principal_name is the service principal name registered with the KDC that identifies the database service.

db_hostname is the host name of the machine running the database.

Specifying User Credentials with Kerberos Authentication

By default, when Kerberos authentication is used, Stylus XQuery takes advantage of the user name and password maintained by the operating system to authenticate users to the database. By allowing the database to share the user name and password used for the operating system, users with a valid operating system account can log into the database without supplying a user name and password.

There may be times when you want to use another set of user credentials. For example, many application servers or Web servers act on behalf of the client user logged on the machine on which the application is running, rather than the server user.

If you want to use user credentials other than the server user's operating system credentials, include code in your application to obtain and pass a `javax.security.auth.Subject` used for authentication as shown in the following example.

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;

import javax.xml.xquery.*;
import com.ddtek.xquery.xqj.DDXQDataSource;

// The following code creates a javax.security.auth.Subject instance
// used for authentication. Refer to the Java Authentication
// and Authorization Service documentation for details on using a
// LoginContext to obtain a Subject.

LoginContext lc = null;
Subject subject = null;

try {
    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}
```

```

    }
    catch (Exception le) {
        ... // display login error
    }

    DDXQDataSource xqDataSource = new DDXQDataSource();

    // This application passes the javax.security.auth.Subject
    // to the driver by executing the driver code as the subject

    XQConnection con =
        (XQConnection) Subject.doAs(subject, new PrivilegedExceptionAction() {
            public Object run() {
                XQConnection con = null;
                try {
                    xqDataSource.setProperty(DDXQDataSource.JDBCURL,
                        "jdbc:xquery:db2://myServer:50000;databaseName=jdbc");
                    con = xqDataSource.getConnection();
                }
                catch (Exception except) {
                    ... //log the connection error
                    return null;
                }

                return con;
            }
        });

    // This application now has a connection that was authenticated with
    // the subject. The application can now use the connection.

    XQExpression xqExpression = con.createExpression();
    String xquery = "for $holding in collection('holdings')/holdings
                    return $holding";
    XQSequence xqSequence = xqExpression.executeQuery(xquery);

    ... // do something with the results

```

Obtaining a Kerberos Ticket Granting Ticket

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a Windows client, the application user does not need to explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

If the application uses Kerberos authentication from a UNIX or Linux client, the user must explicitly obtain a TGT. To explicitly obtain a TGT, the user must log onto the Kerberos server using the `kinit` command. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

```
kinit -l 10h -r 5d user
```

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the `kinit` command and obtaining TGTs for users.

Using NTLM Authentication

NTLM authentication is a single sign-on OS authentication method. This method provides authentication from Windows clients only and requires minimal configuration.

Stylus XQuery supports NTLM authentication for the following databases:

- Oracle
- Microsoft SQL Server

Verify that your environment meets the requirements listed in [Table 7-3](#) before you configure the driver for NTLM authentication.

Table 7-3. NTLM Authentication Requirements

Component	Requirements
Database server	<p>The database server must be administered by the same domain controller that administers the client and must be running one of the following databases:</p> <p>Oracle:</p> <ul style="list-style-type: none"> ■ Oracle 11g ■ Oracle 10g (R1 and R2) ■ Oracle 9i (R1 and R2) <p>Microsoft SQL Server:</p> <ul style="list-style-type: none"> ■ Microsoft SQL Server 2008 ■ Microsoft SQL Server 2005 ■ Microsoft SQL Server 2000 Service Pack 3 or higher ■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
Domain controller	<p>The domain controller must administer both the database server and the client. Network authentication must be provided by NTLM on one of the following operating systems:</p> <ul style="list-style-type: none"> ■ Windows Server 2003 ■ Windows 2000 Server Service Pack 3 or higher

Table 7-3. NTLM Authentication Requirements

Component	Requirements
Client	<p>The client must be administered by the same domain controller that administers the database server and must be running on one of the following operating systems:</p> <ul style="list-style-type: none">■ Windows Vista■ Windows Server 2003■ Windows XP Service Pack 1 or higher■ Windows 2000 Service Pack 4 or higher■ Windows NT 4.0

Configuring NTLM Authentication

Stylus XQuery provides three NTLM authentication DLLs:

- DDJDBCAuthxx.dll (32-bit)
- DDJDBC64Authxx.dll (Itanium 64-bit)
- DDJDBCx64Authxx.dll (AMD64 and Intel EM64T 64-bit)

where *xx* is a two-digit number.

The DLLs are located in the *install_dir/lib* directory (where *install_dir* is your Stylus XQuery installation directory). If the application using NTLM authentication is running in a 32-bit JVM, Stylus XQuery automatically uses DDJDBCAuthxx.dll. Similarly, if the application is running in a 64-bit JVM, DDJDBC64Authxx.dll or DDJDBCx64Authxx.dll is used.

To configure Stylus XQuery:

- 1 Set the AuthenticationMethod connection property to auto (the default) or ntlm. See the Oracle and Microsoft SQL Server connection properties tables in [“Database Connection Properties” on page 460](#) for more information about setting a value for this property.
- 2 By default, Stylus XQuery looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install Stylus XQuery in a directory that is not on the Windows system path, perform one of the following actions to ensure the DDLs can be loaded:

- Add the *install_dir/lib* directory to the Windows system path, where *install_dir* is the Stylus XQuery installation directory.
- Copy the NTLM authentication DLLs from *install_dir/lib* to a directory that is on the Windows system path, where *install_dir* is the Stylus XQuery installation directory.
- Set the LoadLibraryPath connection property to specify the location of the NTLM authentication DLLs. For example, if you install Stylus XQuery in a directory named "StylusXQuery" that is not on the Windows system path, you can use the LoadLibraryPath connection property to specify the directory containing the NTLM authentication DLLs. For example, for SQL Server:

```
jdbc:xquery:sqlserver://server3:1521;  
databaseName=test;LoadLibraryPath=C:\StylusXQuery\lib;  
User=test;Password=secret
```

See the database connection properties tables in [“Specifying Connection URIs” on page 141](#) for more information about setting a value for this property.

- 3 If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow Stylus XQuery to establish connections. See

[“Permissions for Establishing Connections” on page 162](#) for an example.

Permissions for Establishing Connections

Using Stylus XQuery on a Java 2 Platform with the standard Security Manager enabled requires certain permissions to be set in the security policy file of the Java 2 Platform. This security policy file can be found in the `jre/lib/security` subdirectory of the Java 2 Platform installation directory.

NOTE: Web browser applets running in the Java 2 plug-in are always running in a JVM with the standard Security Manager enabled.

To run an application on a Java 2 Platform with the standard Security Manager, use the following command:

```
java -Djava.security.manager application_class_name
```

where *application_class_name* is the class name of the application.

Refer to your Java 2 Platform documentation for more information about setting permissions in the security policy file.

To establish a connection to the database server, Stylus XQuery must be granted the permissions as shown in the following example:

```
grant codeBase "file:/install_dir/lib/-" {
    permission java.net.SocketPermission "*", "connect";
};
```

where *install_dir* is the Stylus XQuery installation directory.

In addition, if Microsoft SQL Server named instances are used, permission must be granted for the listen and accept actions as shown in the following example:

```
grant codeBase "file:/install_dir/lib/-" {  
    permission java.net.SocketPermission "*", "listen, connect, accept";  
};
```

where *install_dir* is the Stylus XQuery installation directory.

Data Encryption Across the Network

If your database connection is not configured to use data encryption, data is sent across the network in a format that is designed for fast transmission. This format does not provide complete protection from hackers, and it can be decoded given some time and effort.

To address data security concerns, you might want to use *data encryption* to provide a more secure transmission of data. Consider using data encryption in the following scenarios:

- You have offices that share confidential information over an intranet.
- You send sensitive data, such as credit card numbers, over a database connection.
- You need to comply with government or industry privacy and security requirements.

NOTE: Data encryption can adversely affect performance because of the additional overhead (mainly CPU usage) required to encrypt and decrypt data.

This section covers the following topics:

- [“Supported Encryption Methods” on page 164](#)
- [“Database-Specific Data Encryption” on page 165](#)
- [“SSL Encryption” on page 166](#)
- [“Configuring SSL for DB2” on page 170](#)
- [“Configuring SSL for Oracle” on page 171](#)
- [“Configuring SSL for Microsoft SQL Server” on page 172](#)
- [“Configuring SSL for Sybase” on page 175](#)

Supported Encryption Methods

Stylus XQuery supports the following encryption methods:

- Database-specific encryption. DB2 defines its own encryption protocol for DB2 for Linux/UNIX/Windows and DB2 for z/OS only. See [“Database-Specific Data Encryption” on page 165](#) for more information.
- Secure Sockets Layer (SSL). SSL is an industry-standard protocol for sending encrypted data over database connections. SSL secures the integrity of your data by encrypting information and providing client/server authentication. See [“SSL Encryption” on page 166](#) for more information.

[Table 7-4](#) summarizes the data encryption methods supported by Stylus XQuery.

Table 7-4. Data Encryption Methods Supported by Stylus XQuery		
Driver	Database-Specific	SSL
DB2 for Linux/UNIX/Windows	X	X ^a
DB2 for z/OS	X	X ^b

Table 7-4. Data Encryption Methods Supported by Stylus XQuery

Driver	Database-Specific	SSL
DB2 for iSeries		X ^c
Informix		
MySQL		
Oracle		X
Microsoft SQL Server		X ^d
Sybase		X

a. Supported for V9.1 Fixpack 2 and higher for Linux/UNIX/Windows.
 b. Supported for DB2 v9.1 for z/OS.
 c. Supported for DB2 V5R3 and higher for iSeries.
 d. Supported for Microsoft SQL Server 2000 and higher.

Database-Specific Data Encryption

The DB2 driver supports a proprietary data encryption protocol for the following DB2 databases:

- DB2 for Linux/UNIX/Windows
- DB2 for z/OS

Configuring Data Encryption for DB2

To configure data encryption for a DB2 database:

- 1 Set the AuthenticationMethod property to clearText, encryptedPassword, or encryptedUIDPassword.
- 2 Set the EncryptionMethod property to DBEncryption or RequestDBEncryption.

SSL Encryption

SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. SSL negotiates the terms of the encryption in a sequence of events known as the *SSL handshake*. The handshake involves the following types of authentication:

- *SSL server authentication* requires the server to authenticate itself to the client.
- *SSL client authentication* is optional and requires the client to authenticate itself to the server after the server has authenticated itself to the client.

NOTE: DB2 and Oracle are the only databases supported by DataDirect Connect *for* JDBC that support SSL client authentication.

The version of SSL that is used and which SSL cryptographic algorithm is used depends on which JVM you are using. Refer to your JVM documentation for more information about its SSL support.

Procedures for configuring SSL vary for the databases that support it. See the individual driver chapters for details about configuring SSL:

- DB2 – [“Configuring SSL for DB2” on page 170](#).
- Oracle – [“Configuring SSL for Oracle” on page 171](#)
- Microsoft SQL Server – [“Configuring SSL for Microsoft SQL Server” on page 172](#)
- Sybase – [“Configuring SSL for Sybase” on page 175](#)

SSL Server Authentication

When the client makes a connection request, the server presents its public certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted

Certificate Authorities (CAs) that resides in an encrypted file on the client known as a *truststore*. Optionally, the client may check the subject (owner) of the certificate. If the certificate matches a trusted CA in the truststore (and the certificate's subject matches the value that the application expects), an encrypted connection is established between the client and server. If the certificate does not match, the connection fails and the driver throws an exception.

To check the issuer of the certificate against the contents of the truststore, the driver must be able to locate the truststore and unlock the truststore with the appropriate password. You can specify truststore information in either of the following ways:

- Specify values for the Java system properties `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`. For example:

```
java -Djavax.net.ssl.trustStore=
C:\Certificates\MyTruststore
```

and

```
java -Djavax.net.ssl.trustStorePassword=
MyTruststorePassword
```

This method sets values for all SSL sockets created in the JVM.

- Specify values for the connection properties `TrustStore` and `TrustStorePassword`. For example:

```
TrustStore=C:\Certificates\MyTruststore
```

and

```
TrustStorePassword=MyTruststorePassword
```

Any values specified by the `TrustStore` and `TrustStorePassword` properties override values specified by the Java system properties. This allows you to choose which truststore file you want to use for a particular connection.

Alternatively, you can configure the DataDirect Connect *for* JDBC drivers to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. If the driver is configured to trust any certificate sent from the server, the issuer information in the certificate is ignored.

SSL Client Authentication (DB2 and Oracle)

If the server is configured for SSL client authentication, the server asks the client to verify its identity after the server has proved its identity. Similar to SSL server authentication, the client sends a public certificate to the server to accept or deny. The client stores its public certificate in an encrypted file known as a *keystore*.

The driver must be able to locate the keystore and unlock the keystore with the appropriate keystore password. Depending on the type of keystore used, the driver also may need to unlock the keystore entry with a password to gain access to the certificate and its private key.

Stylus XQuery can use the following types of keystores:

- Java Keystore (JKS) contains a collection of certificates. Each entry is identified by an alias. The value of each entry is a certificate and the certificate's private key. Each keystore entry can have the same password as the keystore password or a different password. If a keystore entry has a password different than the keystore password, the driver must provide this password to unlock the entry and gain access to the certificate and its private key.
- PKCS #12 keystore contains only one certificate. To gain access to the certificate and its private key, the driver must provide only the keystore password. The file extension of the keystore must be .pfx or .p12.

You can specify this information in either of the following ways:

- Specify values for the Java system properties `javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`. For example:

```
java -Djavax.net.ssl.keyStore=
C:\Certificates\MyKeystore
```

and

```
java -Djavax.net.ssl.keyStorePassword=
MyKeystorePassword
```

This method sets values for all SSL sockets created in the JVM.

NOTE: If the keystore specified by the `javax.net.ssl.keyStore` Java system property is a JKS and the keystore entry has a password different than the keystore password, the `KeyPassword` connection property must specify the password of the keystore entry. For example:

```
KeyPassword=MyKeyPassword
```

- Specify values for the connection properties `KeyStore` and `KeyStorePassword`. For example:

```
KeyStore=C:\Certificates\MyKeyStore
```

and

```
KeyStorePassword=MyKeystorePassword
```

NOTE: If the keystore specified by the `KeyStore` connection property is a JKS and the keystore entry has a password different than the keystore password, the `KeyPassword` connection property must specify the password of the keystore entry. For example:

```
KeyPassword=MyKeyPassword
```

Any values specified by the `KeyStore` and `KeyStorePassword` properties override values specified by the Java system

properties. This allows you to choose which keystore file you want to use for a particular connection.

Configuring SSL for DB2

The DB2 driver supports SSL encryption for the following databases:

- DB2 V9.1 Fixpack 2 and higher for Linux/UNIX/Windows
- DB2 v9.1 for z/OS
- DB2 V5R3 and higher for iSeries

NOTE: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the `LoginTimeout` property to avoid problems when connecting to a server that does not support SSL.

To configure SSL encryption:

- 1 Set the `EncryptionMethod` property to `SSL`.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the `TrustStore` and `TrustStore` properties or their corresponding Java system properties (`javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`, respectively).
- 3 To validate certificates sent by the database server, set the `ValidateServerCertificate` property to `true`.
- 4 Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

- 5 If your database server is configured for SSL client authentication, configure your keystore information:
 - a Specify the location and password of the keystore file. Either set the KeyStore and KeyStore properties or their corresponding Java system properties (javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword, respectively).
 - b If any key entry in the keystore file is password-protected, set the KeyPassword property to the key password.

Configuring SSL for Oracle

The Oracle driver supports SSL encryption for the following databases:

- Oracle 11g (R1)
- Oracle 10g (R1 and R2)
- Oracle 9i (R1 and R2)

Oracle Advanced Security must be enabled on the database server to support SSL.

NOTE: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the LoginTimeout property to avoid problems when connecting to a server that does not support SSL.

To configure SSL encryption:

- 1 Set the EncryptionMethod property to SSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).

- 3 To validate certificates sent by the database server, set the `ValidateServerCertificate` property to `true`.
- 4 Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.
- 5 If your database server is configured for SSL client authentication, configure your keystore information:
 - a Specify the location and password of the keystore file. Either set the `KeyStore` and `KeyStorePassword` properties or their corresponding Java system properties (`javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`, respectively).
 - b If any key entry in the keystore file is password-protected, set the `KeyPassword` property to the key password.

Configuring SSL for Microsoft SQL Server

The SQL Server driver supports SSL encryption for the following databases:

- Microsoft SQL Server 2000 or higher
- Microsoft SQL Server 2000 Enterprise Edition (64-bit) or higher

Depending on your Microsoft SQL Server configuration, you can choose to encrypt all data, including the login request, or encrypt the login request only. Encrypting login requests, but not data, is useful for the following scenarios:

- When your application needs security, but cannot afford to pay the performance penalty for encrypting data transferred between the driver and server.
- When the server is not configured for SSL, but your application still requires a minimum degree of security. (Applicable to Microsoft SQL Server 2005 and higher only.)

NOTE: When SSL is enabled, the driver communicates with database protocol packets set by the server's default packet size. Any value set by the PacketSize property is ignored.

Using SSL with Microsoft SQL Server

If your Microsoft SQL Server database server has been configured with an SSL certificate signed by a trusted CA, the server can be configured so that SSL encryption is either optional or required. When required, connections from clients that do support SSL encryption fail.

Although a signed trusted SSL certificate is recommended for the best degree of security, Microsoft SQL Server 2005 and higher can provide limited security protection even if an SSL certificate has not been configured on the server. If a trusted certificate is not installed, the server will use a self-signed certificate to encrypt the login request, but not the data.

Table 7-5 shows how the different EncryptionMethod property values behave with different Microsoft SQL Server configurations.

Table 7-5. EncryptionMethod Property and Microsoft SQL Server Configurations

Value	No SSL Certificate	SSL Optional	SSL Required
noEncryption	Login request and data are not encrypted.	Login request and data are not encrypted.	Connection attempt fails.

Table 7-5. EncryptionMethod Property and Microsoft SQL Server Configurations

Value	No SSL Certificate	SSL Optional	SSL Required
SSL	Connection attempt fails.	Login request and data are encrypted.	Login request and data are encrypted.
requestSSL	Login request and data are not encrypted.	Login request and data are encrypted.	Login request and data are encrypted.
loginSSL	Microsoft SQL Server 2005 and higher: Login request is encrypted; data is not. Microsoft SQL Server 2000: Connection attempt fails.	Login request is encrypted; data is not.	Login request and data are encrypted.

How to Configure SSL for Microsoft SQL Server

To configure SSL encryption for Microsoft SQL Server:

- 1 Choose the type of encryption for your application:
 - If you want the driver to encrypt all data, including the login request, set the EncryptionMethod property to SSL or requestSSL.
 - If you want the driver to encrypt only the login request, set the EncryptionMethod property to loginSSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.

- 4 Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

Configuring SSL for Sybase

The Sybase driver supports SSL encryption for the following databases:

- Sybase Adaptive Server Enterprise 15.0
- Sybase Adaptive Server Enterprise 12.5, 12.5.1, 12.5.2, 12.5.3, and 12.5.4

In addition, the Sybase Security and Directory Services package, `ASE_SECDIR`, is required.

NOTE: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the `LoginTimeout` property to avoid problems when connecting to a server that does not support SSL.

To configure SSL encryption:

- 1 Set the `EncryptionMethod` property to `SSL`.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the `TrustStore` and `TrustStore` properties or their corresponding Java system properties (`javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`, respectively).
- 3 To validate certificates sent by the database server, set the `ValidateServerCertificate` property to `true`.

- 4 Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

8 Improving Performance

This chapter provides information and techniques you can use to enhance the performance of your Stylus XQuery applications. This chapter covers the following topics:

- [“Querying Large XML Documents” on page 177](#)
- [“Using Comparisons” on page 189](#)
- [“Understanding Compensation” on page 191](#)
- [“Using Query Pooling” on page 192](#)
- [“Using Connection Pooling” on page 193](#)

If you work with relational databases, see [Stylus XQuery SQL Generation Algorithms” on page 260](#) to learn about different ways Stylus XQuery can generate translations of XQuery code to SQL statements.

Querying Large XML Documents

Querying large XML documents can present processing challenges, both in terms of query performance and memory resources. The Stylus XQuery Streaming XML feature provides an efficient way to process XQuery, especially against large documents.

This section describes what the Streaming XML feature is, how to use it, and provides several examples. It covers the following topics:

- [“What is Streaming XML?” on page 178](#)
- [“Enabling Streaming XML” on page 178](#)
- [“Data Sources” on page 180](#)
- [“Using Plan Explain” on page 181](#)

- [“Taking Advantage of Streaming XML” on page 182](#)
- [“Streaming XML Examples” on page 185](#)

What is Streaming XML?

The Stylus XQuery engine supports a processing technique known as *Streaming XML*. Streaming XML processes a document sequentially, discarding portions of the document that are no longer needed to produce further query results. This technique reduces memory usage because only the portion of a document needed at a given stage of query processing is instantiated in memory – it simultaneously parses the XML document, executes the query, and sends the data to the application as needed.

The Streaming XML feature operates on a per XML document basis. For example, in a single query, the Streaming XML feature might be used for XML document A and not for XML document B. See [“Streaming XML Is Not Always Used” on page 179](#) for more information on this topic.

Enabling Streaming XML

The Streaming XML feature is enabled by default. You can override the default behavior in one of two ways:

- Set `ddtek:xml-streaming="no"` in the query prolog. See [“Using Option Declarations and Extension Expressions” on page 275](#) for more information on this topic.
- Set the `"streaming"` attribute in the `<request>` element to `no`. See [“HTTP Functions <request> Element” on page 433](#) for more information on this topic.

Streaming XML Is Not Always Used

When Streaming XML is enabled, the Stylus XQuery engine makes the determination to use it when the XQuery is executed. There are certain circumstances, however, in which Streaming XML is not used, even if it is enabled:

- If the XML document possibly needs to be parsed more than once. For example, the following two circumstances require an XML document to be parsed more than once:
 - If the query includes `fn:doc()` without literal arguments. In this case, the documents to be queried are determined at runtime and, therefore, might be parsed twice.
 - If the `fn:doc()` expression is used in multiple for clauses in a FLWOR expression. For example:

```
for $a in doc("A.XML")/A/B/C
return
  for $b in doc("X.XML")/X/Y/Z
  return
    ...
```

In this case, Streaming XML is used for A.XML, but not for X.XML.

- If nodes from the XML document are accessed with a reverse or optional axis, or with any function that is based indirectly on such an axis: `fn:root()`, `fn:id()`, and `fn:idref()`.

When Streaming XML is not used, the Stylus XQuery engine loads the entire XML document in memory and creates an optimized in-memory representation of it. The in-memory representation is used during query execution and then discarded. In general, this technique requires more memory than Streaming XML, but it can be more efficient (in terms of processing time) for certain XQuery.

Streaming Can Be Interrupted

In the following circumstances, some expressions can cause the Streaming XML feature to stop processing the current node:

- A node is used in a function or operator, including effective boolean value calculations. For example:

```
...
if (doc ("foo.xml") /a/b/c
...

```

In this example, Streaming XML is used for the a and b nodes, but the c nodes and all of its children are instantiated in memory.

- Multiple path expressions are evaluated on a node. For example:

```
...
doc ("foo.xml") /a/b/ (c|d)
...

```

In this example, Streaming XML is used for the a nodes, but the b nodes and all of its children are instantiated in memory.

- A node is referenced multiple times in the query.

You can easily see whether or not Streaming XML is being used to process an XQuery using Stylus XQuery Plan Explain. See [“Using Plan Explain” on page 181](#) for more information.

Data Sources

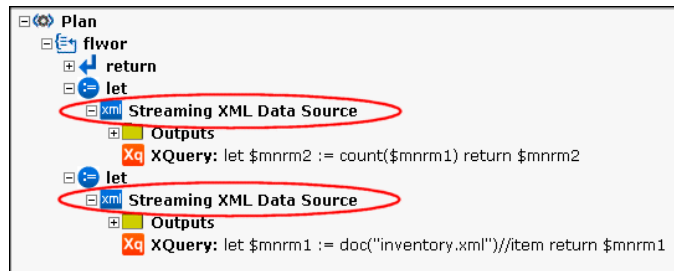
Stylus XQuery supports Streaming XML on XML documents accessed through:

- `fn:doc()`
- `fn:collection()` when using the extensions that allow querying of directories (see [“Querying Multiple Files in a Directory” on page 288](#))

- External variables and initial context item. For the input values to be streamed, they must be defined as XQSequence, java.io.Reader, java.io.InputStream, StAX, or a SAXSource containing only an InputSource property (that is, not XMLReader). For streaming to work with external variables when not using prepared queries, Stylus XQuery must be using deferred binding (see [“Support of Deferred Binding” on page 532](#)).
- External Java functions. For the return values to be streamed, they must be defined as javax.xml.transform.stream.StreamSource, javax.xml.transform.stax.StAXSource (for JVM 1.6 only), or com.ddtek.xquery.StAXSource.

Using Plan Explain

Plan Explain allows you to generate an XQuery execution plan that lets you see how Stylus XQuery will execute your query. Among other information about your XQuery, Plan Explain shows you whether or not the Stylus XQuery engine will use Streaming XML, as shown in the following illustration:



See [“Generating XQuery Execution Plans” on page 307](#) to learn more about Plan Explain.

Taking Advantage of Streaming XML

Depending on the task performed by your XQuery, it is possible to make small changes to your XQuery to take advantage of the performance benefits provided by Streaming XML.

Working with XML Headers

Streaming XML can be useful when parts of an input document are used to create a header in the result, and numerous transformations are performed on the rest of the result. Streaming XML can be especially beneficial when dealing with large input documents.

Consider the following XML document, which lists numerous stock holdings for an individual (imagine `<holding>` elements numbering in the hundreds or even thousands).

```
<?xml version="1.0"?>
  <person>
    <first-name>John</first-name>
    <last-name>Smith</last-name>
    <holdings>
      <holding ticker="PRGS">1000</holding>
      <holding ticker="STOCK1">2000</holding>
      <holding ticker="STOCK2">3000</holding>
      <!-- ... -->
    </holdings>
  </person>
```

Your XQuery needs to create a separate XML document for each stock holding, using the header information to create a `<person>` element and then listing holding information, like this:

```
<person lastName="Smith" name="John">
  <holding ticker="PRGS">1000</holding>
</person>
```

The XQuery used to provide this XML output could look like this:

```

let $firstName := doc("header.xml")/person/first-name
let $lastName := doc("header.xml")/person/last-name
for $holding at $pos in
doc("header.xml")/person/holdings/holding
return
  ddtek:serialize-to-url(
    <person name="{ $firstName}" lastName=
"{ $lastName}">{ $holding}</person>,
    concat("output-", $pos, ".xml"), "indent=yes"
  )

```

In this case, though, the Streaming XML feature is not used where it will provide the most benefit. Indeed, it is used only for minor formatting operations performed on the XQuery output.

Making a simple change to the XQuery (shown in bold in the following code sample) ensures that Streaming XML is used throughout the XQuery – most importantly in the loop formed by the FLWOR expression:

```

let $firstName as element() := doc("header.xml")/person/first-name
let $lastName as element() := doc("header.xml")/person/last-name
for $holding at $pos in doc("header.xml")/person/holdings/holding
return
  ddtek:serialize-to-url(
    <person name="{ $firstName}" lastName="{ $lastName}">{ $holding}</person>,
    concat("output-", $pos, ".xml"), "indent=yes"
  )

```

The `as element()` declarations tell Stylus XQuery that the `first-name` and `last-name` elements in the source document are singletons, which allows the Stylus XQuery engine to use Streaming XML on the FLWOR expression.

Aggregation Functions

XQuery aggregation functions – functions that count elements in an XML document, for example – can take advantage of the efficiencies made available by the Streaming XML feature. Aggregation functions include:

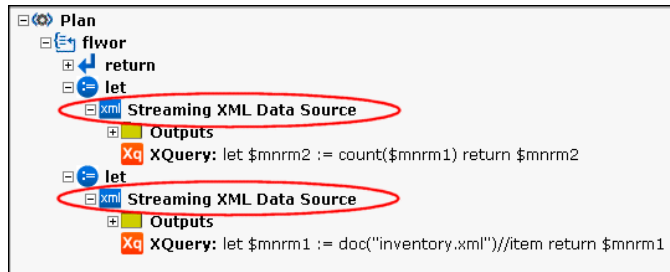
- fcn:count()
- fn:min()
- fn:max()
- fn:sum()
- fn:avg()

Example

Consider the following XQuery; imagine that `inventory.xml` contains thousands of `<item>` elements:

```
count(doc('inventory.xml')//item)
```

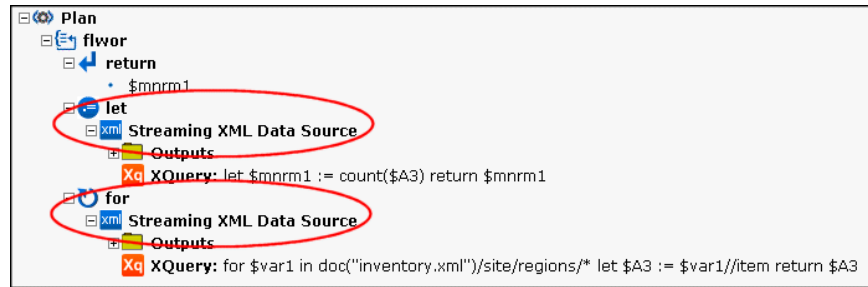
Here, the `count()` function is simply counting the number of `<item>` elements in the `inventory.xml` document. Examining the XQuery using Plan Explain, we can see that Streaming XML is used in two let clauses:



If we make the XQuery slightly more complicated, by returning the number of `<item>` elements per `<region>`:

```
for $b in doc('inventory.xml')/site/regions/*
return count($b//item)
```

XML Streaming is still used to process this XQuery, but note that the XQuery uses a `let-` and `for-` clause, rather than two `let-` clauses, as in the previous example:



Streaming XML Examples

This section provides several examples of the Streaming XML feature, including examples of when it is not used by the Stylus XQuery engine to process the XQuery. The examples are commented, allowing you to easily copy/paste them into test applications.

When Streaming XML Is Used

The following show examples of XQuery in which Streaming XML is used.

Simple Path Expressions

```
(:
A simple path expression.
The complete document can be processed in streaming mode.
:)
doc("file.xml")/a/b/c
```

```
(:
A simple path expression.
The complete document can be processed in streaming mode.
If a c element is a descendent of a parent c element, it is
memorized.
:)
doc("file.xml")/a/b//c
```

Path Expression with Predicate

```
(:
A path expression with predicate.
The document is queried using the Streaming XML feature.
Only the values of d that match the predicate are
materialized; all c's and x's are materialized and
discarded one by one.
:)
doc("file.xml")/a/b/c[x eq 1]/d
```

Path Expression with Attribute Predicate

```
(:
A path expression with attribute predicate.
The document is queried using the Streaming XML feature.
No materialization is performed. Only general comparisons
with attribute tests are supported.
:)
doc("file.xml")//ITEMS[@ITEMNO eq '1004']
```

XQuery Expression with fn:data

```
(:
The document is queried using the Streaming XML feature.
Atomization on streaming results is supported.
ITEMNO elements are not first materialized and then
atomized.
:)
fn:data(doc("file.xml")//ITEMS/ITEMNO
```

XQuery Expression with Function on Node

```
(:
The document is queried using the Streaming XML feature.
Functions on nodes (fn:name(), fn:node-name(),
fn:local-name(), etc.) are supported.
:)
doc("file.xml")//ITEMS/element()[fn:local-name(.)
eq 'ITEMNO']
```

XQuery Expression with exists

```
(:
The document is queried using the Streaming XML feature.
Existential tests are supported.
:)
doc("file.xml")//ITEMS[exists(@ITEMNO)]
doc("file.xml")//ITEMS[exists(ITEMNO)]
doc("file.xml")//ITEMS/ITEMNO[exists(.)]
```

Two XML Documents

```
(:
Two different documents in a sequence. Both are queried
with the Streaming XML feature.
:)
doc("file1.xml")/a/b/c,
doc("file2.xml")/x/y/z
```

Complex Example Using the Streaming XML Feature

```
(:
The document is queried using the Streaming XML feature.
:)
<orders>{
  for $order in doc("orders.xml")//orders
  for $customer in collection("CUSTOMER")/CUSTOMER[CUST_ID = $order/customer]
  return
    <order id="{ $order/@id}">
      <customer>
        <name>{ $customer/CUST_NAME/data(.) }</name>
        <address>{ $customer/CUST_ADDRESS/data(.) }</address>
      </customer>
    </order>
}</orders>
```

```
(:
If the for clauses are switched, the orders.xml document is queried multiple
times; therefore, streaming is not used and the document is instantiated.
:)
```

```

<orders>{
  for $customer in collection("CUSTOMER")/CUSTOMER
  for $order in doc("orders.xml")//orders
  where $customer /CUST_ID = $order/customer
  return
    <order id="{ $order/@id}">
      <customer>
        <name>{$customer/CUST_NAME/data(.)}</name>
        <address>{$customer/CUST_ADDRESS/data(.)}</address>
      </customer>
    </order>
}</orders>

```

When Streaming XML Is Not Used

The following show examples of XQuery in which Streaming XML is not used.

Reverse Axis

```

(:
The Streaming XML feature is not used due to the reverse
axis.
:)
```

```
doc("file.xml")/a/b/c/../d
```

```

(:
This query could have been written as follows, in which
case the b elements are materialized one by one.
:)
```

```
doc("file.xml")/a/b[c]/d
```

Optional Axis

```

(:
The Streaming XML feature is not used due to the
preceding-sibling optional axis.
:)
```

```
doc("file.xml")/a/b[c=5]/preceding-sibling::*[1]
```

Two Documents

```
(:
Two documents, not queried with the Streaming XML feature
as the same document. These documents are possibly queried
twice.
:)
declare variable $file as xs:string external;
doc("file1.xml")/a/b/c,
doc($file)/x/y/z
```

Using Comparisons

When Stylus XQuery encounters comparisons in where clauses or in predicate expressions *and* an operand is bound to data in an XML data source, performance can be significantly improved if this operand is known by Stylus XQuery to be a single item.

Consider the following query:

```
for $request in doc('file:///c:/in/request.xml')/request
let $ticker := $request/performance/ticker,
    $start := $request/performance/start,
    $end := $request/performance/end
for $h in collection('historical')/historical
where $h/ticker = $ticker
return $h
```

Stylus XQuery does not know how many ticker, start, or end elements may occur in the XML source, so it restricts its rewrites in case there are more than one of each of these elements.

Using value comparisons – for example, `eq`, as shown in the following code – instead of general comparisons makes this query run faster:

```
for $request in doc('file:///c:/in/request.xml')/request
let $ticker := $request/performance/ticker,
    $start := $request/performance/start,
    $end := $request/performance/end
for $h in collection('historical')/historical
where $h/ticker eq $ticker
return $h
```

However, this does not work for all data types because `eq` is restrictive in the types it accepts and does less implicit casting.

Generally, using value comparisons (`eq`, `ne`, `lt`, `le`, `gt`, `ge`) instead of general comparisons (`=`, `!=`, `<`, `<=`, `>`, `>=`) improves performance. When using general comparison against sequences, the result of the expression is true if any combination of the items contained in the sequences satisfies the comparison. Value comparison only applies to operands that are single items, and the expression returns true if the single items compared with the value comparison operator (for example, `eq`) match. If one of the two operands is not a single item, the use of `eq` raises an error. In the preceding example, the query behavior perceived by the user does not differ when using `=` or `eq` because `$h/ticker` and `$ticker` are always single items. But, typically, using `eq` instead of `=` significantly improves performance.

Understanding Compensation

XQuery contains expressions, functions, and operators that cannot be directly translated into SQL. For example, `fn:tokenize()` has no SQL equivalent. When an XQuery expression cannot be translated to SQL, Stylus XQuery "compensates" the expression; that is, it executes the expression in the Stylus XQuery Engine using data retrieved from the database. Compensation provides full-featured XQuery functionality, but it is often slower than executing an expression in the database.

Sometimes, the same result can be obtained by using an expression that does not require compensation. For example, suppose you need to perform string comparisons with data that contains trailing spaces. You could use the XQuery function `normalize-space()`, which removes leading and trailing spaces:

```
for $h in collection('stocks.dbo.historical')/historical
where normalize-space($h/ticker) = 'AMZN'
return $h
```

However, the `normalize-space()` function is compensated, which means that the `where` clause is evaluated in the Stylus XQuery engine rather than in the database, which slows performance. As shown in the following example, the most efficient solution is to use the function `rtrim()`, which is available for XML documents and all supported databases:

```
for $h in collection('historical')/historical
where ddtek:rtrim($h/ticker) = 'AMZN'
return $h
```

See [Appendix A "XQuery Support" on page 339](#) for details about which expressions, functions, and operators are compensated.

Using Query Pooling

Query pooling allows an application to reuse queries that have been executed. If your Java application executes the same query more than once, you can improve performance by enabling Stylus XQuery's internal query pooling. When query pooling is enabled, Stylus XQuery caches a specified number of queries executed by an application. Stylus XQuery pools queries executed using XQExpression and XQPreparedExpression.

Using XQJ, you can enable query pooling by specifying the DDXQDataSource MaxPooledQueries property. For example, if the DDXQDataSource MaxPooledQueries property is set to 20, Stylus XQuery caches the last 20 queries executed by the application. If the value set for this property is greater than the number of queries used by the application, all queries are cached. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#) for more information.

Using Connection Pooling

Connection pooling allows your application to reuse connections. Stylus XQuery supports connection pooling through JDBC and supports JDBC connection pool managers in the following application server environments:

- Tomcat 5.x and 6.x
- JBoss 4.x and 5.x
- BEA WebLogic Platform 9.x and 10.x
- IBM WebSphere Application Server V6.1
- Oracle Application Server 10g

For the most current information about which application server environments are supported, go to:

<https://www.xquery.com/connection-pooling>

Stylus XQuery has a JDBC driver that allows your application to use connection pooling with an application server's JDBC pool manager. See [“Configuring Connection Pooling” on page 196](#) for information about this driver.

Stylus XQuery provides three classes for support of connection pooling:

- `com.ddtek.xquery.jdbc.XQueryDriver`. This is the driver class for the Stylus XQuery JDBC driver, which is used to configure a pooled connection through the JDBC Driver Manager. See [“Configuring a Connection Through the JDBC Driver Manager” on page 194](#).
- `com.ddtek.xquery.jdbc.XQueryConnectionPoolDataSource`. This class is used to configure a pooled connection through a data source. See [“Configuring a Connection Through a Data Source” on page 195](#).

- `com.ddtek.xquery.jdbc.XQueryConnection`. This class is used to convert a JDBC connection into an XQJ connection. See the example in [“Configuring Connection Pooling” on page 196](#).

Configuring a Connection Through the JDBC Driver Manager

The `com.ddtek.xquery.jdbc.XQueryDriver` class is an implementation of the JDBC driver interface (`java.sql.Driver`) and is used to configure a pooled connection through the JDBC Driver Manager.

The syntax of the JDBC URL required by this class depends on if you are connecting to one relational data source or multiple relational data sources.

If you are connecting to one relational data source the syntax is:

```
jdbc:datadirect:xquery://JdbcUrl={url}[,optionalProperty=value[;...]]
```

where:

url is a URL, as documented in [“Specifying Connection URIs” on page 141](#).

optionalProperty is any of the `DDXQDataSource` properties, as documented in [Table 6-1, “DDXQDataSource Properties,” on page 128](#).

value is determined by the `DDXQDataSource` property you are specifying.

In the following example, the JDBC URL is defined in the first set of braces `{}` and the `baseURI` is a property of `DDXQDataSource`:

```
jdbc:datadirect:xquery://JdbcUrl={jdbc:xquery:sqlserver://localhost:1433;  
databaseName=holdings;User=myuserID;Password=mypwd};  
baseURI={file:///C:/xml/documents/};
```

If you are connecting to multiple relational data sources the syntax is:

```
jdbc:datadirect:xquery://jdbcConnections={Url={url}[,optionalProperty=value
[;...]]},...
```

where:

url is a URL, as documented in [“Specifying Connection URIs” on page 141](#).

optionalProperty is any of the DDXQJDBCConnection properties, as documented in [Table 6-2, “DDXQJDBCConnection Properties,” on page 136](#).

value is determined by the DDXQJDBCConnection property you are specifying.

In the following example, two JDBC URLs are specified and the user ID and password are specified using the User and Password properties of DDXQJDBCConnection; they are not specified within the URL:

```
jdbc:datadirect:xquery://jdbcConnections=
{Url={jdbc:xquery:sqlserver://server1:1433;databaseName=stocks};User=myuserID;
Password=mypwd}, {Url={jdbc:xquery:sqlserver://server2:1433;databaseName=
holdings};
User=myuserID2;Password=mypwd2}
```

Configuring a Connection Through a Data Source

The `com.ddtek.xquery.jdbc.XQueryConnectionPoolDataSource` class is an implementation of the JDBC interface `javax.sql.ConnectionPoolDataSource` and is used to configure a connection pool connection through a data source.

Some application servers, such as IBM WebSphere, require a data source to configure pooled connections and provide the infrastructure for you to create the `ConnectionPoolDataSource` objects you need to configure connection pooling.

The only property defined for this class is `connectionURL`. The value for this property is a URL, which is documented in [“Configuring a Connection Through the JDBC Driver Manager” on page 194](#).

Configuring Connection Pooling

- 1 The Stylus XQuery JDBC driver must be registered in the application server and a connection pool must be created. This step makes the pooled connections available to your application code through JNDI. For example:

```
Context envContext =
    (Context)initContext.lookup("java:/comp/env");
DataSource jdbc_ds =
    (DataSource)envContext.lookup("jdbc/DDXQExample");
Connection jdbc_c = jdbc_ds.getConnection();
```

Now, a JDBC connection is available.

- 2 Convert the JDBC connection into an XQJ connection. For example:

```
XQConnection xqj_c =
    XQueryConnection.getXQConnection(jdbc_c);
```

NOTE: Stylus XQuery provides a class, `com.dtek.xquery.jdbc.XQueryConnection`, to convert the JDBC connection into an XQJ connection.

Now, an XQJ connection is available and can be used with XQJ. For example:

```
XQPreparedExpression xqj_p =
    xqj_c.prepareExpression("fn:doc('foo.xml')//abc");
```

- 3** To make a connection available again for pooling, you must close the JDBC connection, not the XQJ connection. For example:

```
jdbc_c.close()
```

For information about how to configure Stylus XQuery to use connection pooling in the supported application server environments, go to:

<https://www.xquery.com/connection-pooling>

Example of Servlet Using Connection Pooling

The following code is a fully functional example for a Java Servlet.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQExpression;
import javax.xml.xquery.XQSequence;

import com.ddtek.xquery.jdbc.XQueryConnection;
```

```

/**
 * Servlet example demonstrating the integration with
 * JDBC Connection Pooling
 */
public class DDXQServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        Connection jdbc_c = null;
        XQExpression xqj_e = null;

        try {
            Context initContext = new InitialContext();
            Context envContext = (Context)initContext.lookup("java:/comp/env");
            DataSource jdbc_ds = (DataSource)envContext.lookup("jdbc/DDXQExample");
            jdbc_c = jdbc_ds.getConnection();

            PrintWriter out = response.getWriter();

            XQConnection xqj_c = XQueryConnection.getXQConnection(jdbc_c);

            xqj_e = xqj_c.createExpression();
            XQSequence xqj_s = xqj_e.executeQuery(
                "    <b>Current date: </b>,current-date(),<br/>," +
                "    <b>Current time: </b>,current-time(), " +
                "    <table border='1'> "+
                "        <tr> "+
                "            <th>User</th> "+
                "            <th>Stock</th> "+
                "            <th>Shares</th> "+
                "        </tr> "+
                "        { "+
                "    for $item in collection('holdings')/holdings "+
                "    return "+
                "        <tr> "+
                "            <td>{$item/userid/data(.)}</td> "+
                "            <td>{$item/stockticker/data(.)}</td> "+
                "            <td>{$item/shares/data(.)}</td> "+
                "        </tr> "+
                "        } "+
                "    </table> ");

```

```

        xqj_s.writeSequence(out, new Properties());

        out.close();
    }
    catch(Exception e){
        throw new ServletException(e);
    }
    finally {
        if (xqj_e != null) try{xqj_e.close();} catch (XQException e) {}
        if (jdbc_c != null) try{jdbc_c.close();} catch (SQLException e) {}
    }
}
}

```


9 Building a Web Service

This chapter provides an overview of the XQueryWebService framework and describes how to use it to build a Web service. It covers the following topics:

- [“XQueryWebService Framework Overview” on page 201](#)
- [“XQueryWebService Framework Architecture” on page 204](#)
- [“Example – Employee Lookup” on page 211](#)
- [“Specifying a Database Connection” on page 213](#)
- [“Choosing an Interface for Web Service Access” on page 216](#)
- [“Tools for Testing Web Service Operations” on page 219](#)
- [“Generating WSDL” on page 221](#)
- [“Using WSDL Service References” on page 223](#)

XQueryWebService Framework Overview

XQueryWebService is a framework that allows you to expose an XQuery as a Web service. Implemented as a library for Java classes, XQueryWebService is designed to simplify the design and implementation of Web Servlet Applications. The jar file for this library, `xquerywebservice.jar`, is located in the `\lib` directory where you install Stylus XQuery.

The XQueryWebService framework provides a Servlet implementation to expose as a Web service XQuery stored in a specific directory. Each XQuery exposes one operation; this operation is expressed in the query body through a function that takes the name of the XQuery file without the extension. For example, the file `emp.xquery` provides the `emp` operation. Parameters (external variables) expressed in the XQuery, if any, are reflected in the operation’s prototype.

XQuery modules cannot be published as Web services. A module can be parsed only indirectly when imported by another query.

An XQuery is compiled using lazy evaluation on the first request; after that it is compiled only if the XQuery source on the disk changes.

Third Party Dependencies

The XQueryWebService framework does not require any additional Java library; it relies only on built-in classes, like JAXP to manipulate XML, which are available from Java 1.4.2 and later.

Web Service Interfaces

The [Web Service Description Language \(WSDL\)](#) specification allows a Web service to be exposed through several types of bindings: HTTP GET, HTTP POST and SOAP over HTTP. This section describes these bindings in greater detail.

HTTP GET

The simplest binding is HTTP GET, often described as REST (Representational State Transfer), in which the Web service call is represented by a URL with all its parameters inline. Consider the following URI:

```
http://examples.xquery.com/employee-lookup/emp.xquery?id=A-C71970F
```

Here, `employee-lookup` is the service name, `emp` is the operation name, and what follows after the question mark is a name/value pair of parameters (`id=A-C71970F`).

HTTP GET is very simple to invoke – it is equivalent to accessing an HTML page on a Web server; any internet browser can invoke an XQuery Web service this way. However, only simple types such as strings or numbers can be passed through HTTP GET, limiting this approach to Remote Procedure Calling (RPC) style Web services. Web service responses from a REST binding are always XML fragments.

HTTP POST

HTTP POST allows you to design Web services with sophisticated request messages in the form of XML fragments. The drawback to HTTP POST is that the XQuery language does not provide a standard function to perform this type of binding.

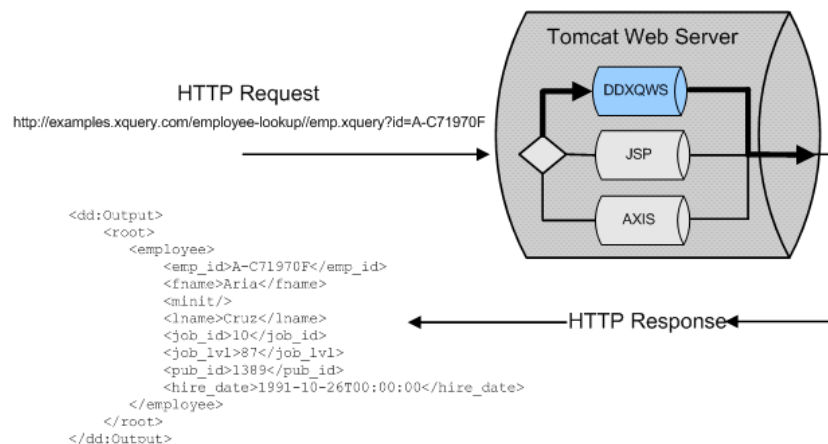
SOAP

SOAP over HTTP is layered on HTTP POST, with the addition of an XML-based wire protocol that describes what the request/response looks like and provides a tighter integration with XML Schema. The SOAP protocol defines an optional element called Header to carry information like user/password or session id.

For more information on Web service interfaces, see
[“XQueryWebService Framework Architecture” on page 204.](#)

XQueryWebService Framework Architecture

A high-level illustration of the XQueryWebService framework architecture looks like this:



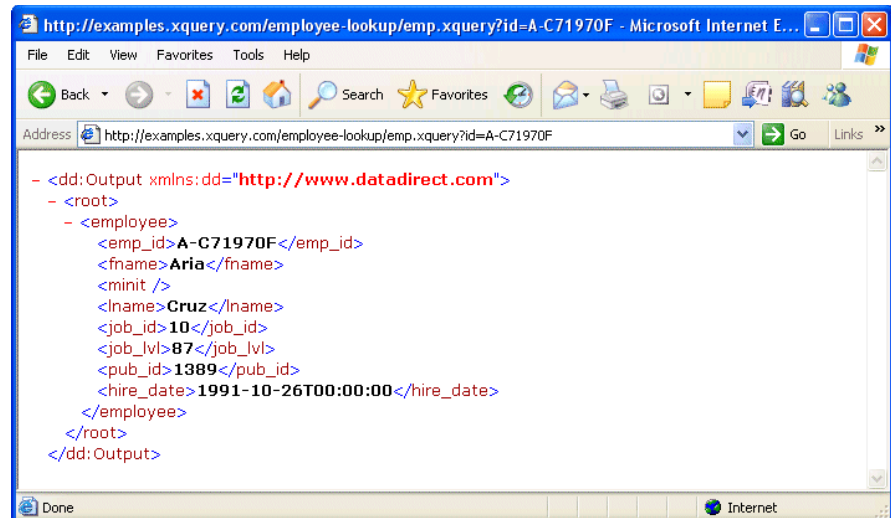
To start, an HTTP request is submitted to a Web server (a Tomcat Web Server in this case). The URI used to invoke the Web service takes the following form:

<http://examples.xquery.com/employee-lookup/emp.xquery?id=A-C71970F>

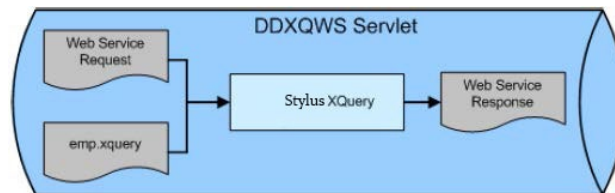
where:

- `http://examples.xquery.com/employee-lookup/emp.xquery` is the location of the XQuery Web service. The Web service was created by saving an XQuery to the `employee-lookup` directory where the Tomcat Web Server is running.
- `id=A-C71970F` is a parameter passed to the XQuery. This parameter, as you will see in a moment, is defined in the XQuery.

When the XQuery is finished, it returns a value using HTTP response, as shown in the following illustration.



Let's take a closer look inside the Stylus XQueryWebService directory on the Web server (DDXQWS).



The browser (or an application) submits the Web service request using SOAP or HTTP GET for the XQuery stored on the Web server. Next, Stylus XQuery unpacks the Web service request and binds its parameters, if any, to the XQuery. In our example, the parameter passed with the Web service request is an ID. The XQuery is then executed and its result (an XML document) is returned to the client.

Example XQuery

To gain a more detailed understanding of what is happening inside the Web service, consider an XQuery, `emp.xquery`. This XQuery retrieves employee data given a unique ID. The query defines a parameter called `id`; the query body is just a single FLWOR expression:

```
declare variable $id as xs:string external;
<root>{
  for $employee in collection("employee")/employee
  where $employee/emp_id = $id
  return $employee
}</root>
```

In the following section, “[Example – Employee Lookup](#)” on [page 211](#), you’ll see how to implement this XQuery as a Web service on your local machine.

The Web Service Description Language (WSDL)

The Web Service Description Language (WSDL) is a language for describing Web services. If we copy the `emp.xquery` to a directory, say `employee-lookup`, where our Java servlet container is running, we can use the following URI to access a WSDL document that describes the Web service that results from our XQuery:

<http://examples.xquery.com/employee-lookup/WSDL>

Using this tool, we can take a closer look at how our XQuery is described by the WSDL document.

Service Element

The service element – only one per WSDL document – is named after the query file name without its extension. The service contains two port definitions that always have the same name: `SOAPPort` and `HTTPGETPort`, respectively; one for SOAP over HTTP, one for HTTP GET.

```
<wsdl:service name="Service">
  <wsdl:port binding="dd:SOAPBinding" name="SOAPPort">
    <wsdlsoap:address
      location="http://examples.xquery.com/employee-lookup/WSDL"/>
  </wsdl:port>
  <wsdl:port binding="dd:HTTPGETBinding" name="HTTPGETPort">
    <http:address
      location="http://examples.xquery.com/employee-lookup/WSDL"/>
  </wsdl:port>
</wsdl:service>
```

Notice that the service address or end point is the same for both ports.

For each element `wsdl:port` under the element `wsdl:service` there is an attribute called `binding=`; the attribute value matches the value of attribute `name=` of one of the `binding` elements.

HTTPGETBinding

The `HTTPGETBinding` describes the HTTP verb (in this case it is GET), which operations are exposed, and how the input/output are encoded. The attribute `location=` in the element `wsdl:operation` is particularly important – it represents the query function to invoke in our query; in this case `emp` means the query body.

```
<wsdl:binding name="HTTPGETBinding" type="dd:HTTPGETPort">
  <http:binding verb="GET"/>
  <wsdl:operation name="emp">
    <http:operation location="/emp"/>
  </wsdl:operation>
</wsdl:binding>
```

```

        <wsdl:input>
            <http:urlEncoded/>
        </wsdl:input>
        <wsdl:output>
            <mime:mimeXML part="Body"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>

```

SOAPBinding

The `SOAPBinding` (in the following code sample) describes the encoding style that will be used by the service; the value can be either `document` or `rpc` (in our case it is always `document`). The style `document` is completely driven by the schema definition associated with the message, so the resulting XML fragment is more elegant. The style `rpc` assumes the creation of a wrapper element that matches the underlying function name to encapsulate the function arguments. The XML on the wire might look the same, but it is conceptually different.

Each `wsdlsoap:operation` defines the attribute `soapAction=` that, similar to the attribute `location=` in `http:operation`, represents the function name; `soapAction=` must be encoded as an HTTP header in the Web service request.

The attribute `use=` in the element `wsdlsoap:body` can be either `literal` or `encoded`. (In the generated WSDL it will be always `literal`, as suggested by the OASIS [WS Basic Profile 1.0](#), to improve interoperability between different client implementations.) The message representation on the wire has the child element of the element `wsdlsoap:body`, which matches the global element defined in the XML Schema and is declared in the related message part.

The attribute `type=` in the element `binding` matches the attribute `name=` of one of the element `portType`. The element `portType` associates one message for the input and one for the output to each operation.

```
<wsdl:binding name="SOAPBinding" type="dd:SOAPPort">
  <wsdlsoap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>
  <wsdl:operation name="emp">
    <wsdlsoap:operation soapAction="emp.xquery" style="document"/>
    <wsdl:input>
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

For each query function there is a pair of messages (input and output) for each binding (`SOAPPort` and `HTTPGETPort`). Having different messages for each binding allows, for instance, simple types like `xs:string` or `xs:integer` to be used for HTTP GET, which can be easily expressed inline as a URI.

```
<wsdl:portType name="SOAPPort">
  <wsdl:operation name="emp">
    <wsdl:input message="dd:empInputMsg"/>
    <wsdl:output message="dd:OutputMsg"/>
    <wsdl:fault name="nmtoken" message="dd:FaultMsg"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:portType name="HTTPGETPort">
  <wsdl:operation name="emp">
    <wsdl:input message="dd:empInputMsg"/>
    <wsdl:output message="dd:OutputMsg"/>
    <wsdl:fault name="nmtoken" message="dd:FaultMsg"/>
  </wsdl:operation>
</wsdl:portType>
```

The element `wsdl:message` may have multiple sub-elements called `wsdl:part`; each `part` references either an XML Schema global type or global element. OASIS [WS Basic Profile 1.0](#) suggests using only one `part` and a global element. To mimic the validation process against an XML Schema, the validation always starts from a global element – the document root.

```
<wsdl:message name="empInputMsg">
  <wsdl:part name="parameters" element="dd:emp"/>
</wsdl:message>

<wsdl:message name="OutputMsg">
  <wsdl:part name="Output" element="dd:Output"/>
</wsdl:message>

<wsdl:message name="FaultMsg"/>
```

Finally, the WSDL describes the element `types` where the XML Schema types are defined. For each message, the XML Schema defines two global elements – one for the input and one for the output.

```
<wsdl:types>
  <wsdl:types>

    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.datadirect.com"
      attributeFormDefault="unqualified"
      elementFormDefault="qualified">
      <xs:import schemaLocation="employee.xsd"
        namespace="http://www.employee.com"/>
      <xs:element name="emp">
        <xs:complexType>
          <xs:all>
            <xs:element type="xs:string" name="id"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element type="xs:anyType" name="Output"/>
    </xs:schema>
  </wsdl:types>
```

Example – Employee Lookup

The application described in the following sections is a simple employee lookup query designed to illustrate some of the features of the XQueryWebService framework. When complete, the query returns information such as first and last name, job level, date of hire, and so on based on the employee ID you enter.

Before You Begin

If you want to run the employee lookup example, you need the following:

- An XQuery editor – An XQuery editor, like [Stylus Studio](#) is needed to author the XQuery you wish to expose as a Web service.
- A Java servlet container – The XQueryWebService framework has been tested with several Java servlet containers, including
 - [Apache Tomcat](#)
 - [JBoss](#)
 - [IBM WebSphere](#)
 - [BEA WebLogic](#)

You can use any Java servlet container you like; we used the open source Apache Tomcat for this example. If you plan to run the example on a different servlet container, you will need to follow its specific deployment procedure.

Setting Up

Once you have your XQuery editor and Java servlet container, you can set up the files needed to write your own employee lookup XQueryWebService framework application. Here's how to get started:

- 1 Copy the all Stylus XQuery jar files to your Java servlet container \lib directory (<Tomcat_dir>\lib, for example). The jar files are located in the \lib directory where you install Stylus XQuery.
- 2 Create an employee-lookup directory under the Java servlet container \webapps directory (<Tomcat_dir>\webapps\employee-lookup, for example).
- 3 Create a WEB-INF directory under the newly created \employee-lookup directory (<Tomcat_dir>\webapps\employee-lookup\WEB-INF, for example).
- 4 Create the following configuration file as web.xml and save it to the WEB-INF directory:

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/
    j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <description>Employee lookup</description>
  <display-name>Employee-lookup</display-name>
```

```

<servlet>
  <servlet-name>XQueryWebService</servlet-name>
  <servlet-class>com.ddtek.xquery.webservice.XQServlet</servlet-class>
  <init-param>
    <param-name>JNDIDataSourceName</param-name>
    <param-value>jdbc/employee-lookup</param-value>
  </init-param>
</servlet>

<resource-ref>
  <res-ref-name>jdbc/employee-lookup</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

<servlet-mapping>
  <servlet-name>XQueryWebService</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>

```

Next Steps

Once you have taken care of the basics, you can begin specifying relational data sources, as described in the following section.

Specifying a Database Connection

If your queries access relational data, you need to register the database connection with the Java servlet container. The XQueryWebService supports

- Single database connections
- Database connection pooling

Specifying a Single Connection

You can use the Web Application configuration file (web.xml) to specify the database connection. This example specifies a connection to Microsoft SQL Server:

```
<init-param>

  <param-name>DDXQJDBCCConnection1</param-name>
  <param-value>jdbc:xquery:sqlserver://localhost;user=sa;DatabaseName=pubs
    </param-value>
</init-param>
```

The name of the `<param-name>` element can be any string you like.

Single database connections are created and then discarded with each request, which can add to an application's processing overhead. A more efficient technique is database connection pooling, which is discussed next.

Database Connection Pooling

Connection pooling is a technique for specifying database connections that allows a Web application to create a database connection on demand, and then return it to a pool when it is no longer need, rather than discarding it. Connection pooling can improve response time and help preserve database resources. Web server requests are locked if no connection is available in the pool.

Another benefit of using connection pooling is that it allows for connection recovery in the event that the connection is lost – if the server times out, for example. Dropped or disrupted connections are automatically replaced once the server is returned to service.

All popular Java servlet containers offer a connection pooling framework, and Stylus XQuery can be plugged into most of them (Apache Tomcat, BEA WebLogic, IBM WebSphere, JBoss,

and Oracle for example). See [“Using Connection Pooling” on page 193](#) for more information.

Creating a Connection Pool – Example

Here's how to create a connection pool in Apache Tomcat:

- 1 Create a META-INF directory under the \employee-lookup directory (<Tomcat_dir>\webapps\employee-lookup\META-INF, for example).
- 2 Place the following configuration file, context.xml, in that directory:

```
<Context path="/employee-lookup" docBase="employee-lookup"
  crossContext="false" reloadable="true" debug="0">

<Resource name="jdbc/employee-lookup"
  auth="Container"
  type="javax.sql.DataSource"
  username="root"
  password="sa"
  driverClassName="com.ddtek.xquery3.jdbc.XQueryDriver"
  url="jdbc:datadirect:xquery3://JdbcUrl=
    {jdbc:mysql://localhost:3306/pubs_dbo?}"
  initialSize="1"
  accessToUnderlyingConnectionAllowed="true"
  validationQuery="SELECT * FROM users"/>
</Context>
```

Note that the `name=` attribute of the `<Resource>` element has to match the `<res-ref-name>` element (here it is `"jdbc/employee-lookup"`) in the `web.xml` configuration file we described previously. This is the name that the Java servlet uses to perform the Java Naming and Directory Interface (JNDI) lookup required to retrieve the connection pool.

Creating a Connection Pool for Other Servers

As mentioned previously, Stylus XQuery supports several other servers in addition to Tomcat, including JBoss, BEA WebLogic, IBM WebSphere Application Server, and Oracle Application Server. You can learn more about support for connection pools for these servers here: [“Using Connection Pooling” on page 193](#).

Next Steps

Once you have specified your relational database connection, you can start to think about the technology you want to use to access the Web services based on your XQuery. Two popular technologies – SOAP and REST – are described in the following section.

Choosing an Interface for Web Service Access

Data services – that is, your XQuery exposed as a Web service – deployed on the XQueryWebService framework can be accessed using two techniques:

- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)

SOAP is a [W3C Recommendation](#) and has been around for nearly a decade. SOAP is usually the more appropriate of the two techniques for complex processing or when security (exposing sensitive data) is an issue. But REST is gaining popularity for a couple of reasons, including minimal requirements on the client, and an interface – the URI – that is straightforward and well-understood.

Sample XQuery

Let's take a look at `emp.xquery`, which we have saved to our local XQuery directory (`c:\MyQueryDir`, as defined in `web.xml`).

```
declare variable $id as xs:string external;

<root>
{
  for $employee in collection("pubs.dbo.employee")/employee
  where $employee/emp_id = $id
  return $employee
}
</root>
```

When run against the SQL Server pubs sample database, this XQuery returns an employee record given an ID ("A-C71970F").

Using REST

Using REST, this XQuery can be executed from any Internet browser using just this URI:

<http://examples.xquery.com/employee-lookup/emp.xquery?id=A-C71970F>

Notice that with REST, the employee ID ("id=A-C71970F") is visible in the URI.

The result looks something like this:

```
<dd:Output xmlns:dd="http://www.datadirect.com">
  <root>
    <employee>
      <emp_id>A-C71970F</emp_id>
      <fname>Aria</fname>
      <minit/>
      <lname>Cruz</lname>
      <job_id>10</job_id>
      <job_lvl>87</job_lvl>
      <pub_id>1389</pub_id>
```

```

        <hire_date>1991-10-26T00:00:00</hire_date>
    </employee>
</root>
</dd:Output>

```

Using SOAP

Using SOAP, on the other hand, requires submitting the following SOAP request (XML):

```

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body>
    <dd:emp xmlns:dd="http://www.datadirect.com">
      <dd:id>A-C71970F</dd:id>
    </dd:emp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The result, shown here, is pretty much the same as the one returned using REST, only now it is “wrapped” in the SOAP envelope:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body>
    <dd:Output xmlns:dd="http://www.datadirect.com">
      <root>
        <employee>
          <emp_id>A-C71970F</emp_id>
          <fname>Aria</fname>
          <minit>
          </minit>
          <lname>Cruz</lname>
          <job_id>10</job_id>
          <job_lvl>87</job_lvl>
          <pub_id>1389</pub_id>
          <hire_date>1991-10-26T00:00:00</hire_date>
        </employee>
      </root>
    </dd:Output>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```

</dd:Output>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

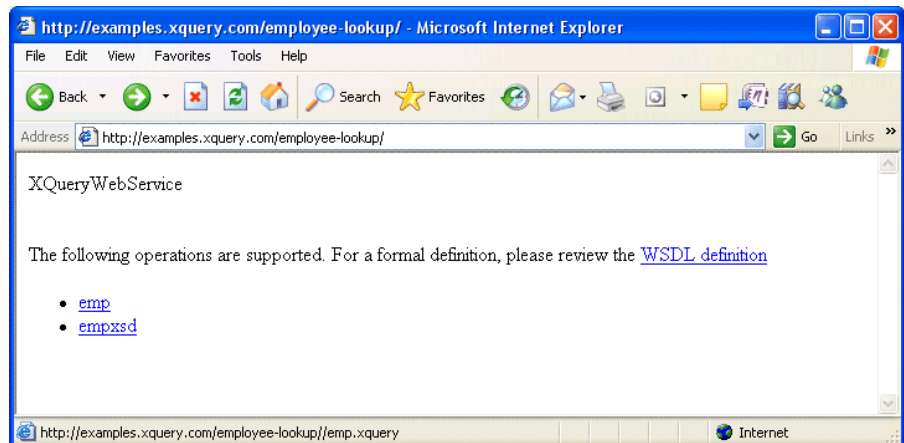
```

Next Steps

The XQueryWebService framework includes some simple tools that let you test the Web service operations you include in your applications. These tools are covered in the next section.

Tools for Testing Web Service Operations

The XQueryWebService framework dynamically lists all the operations exposed by the Web service created from the XQuery in your Java servlet container's XQuery directory. This page, for example, was generated by the Employee Lookup example:



To display this page, just point your browser to the XQueryWebService root – <http://examples.xquery.com/employee-lookup/>, for example.

This HTML form is dynamically created by parsing the XQuery. Clicking one of the exposed Web service operations (in our example, emp) displays another HTML form you can use to provide values for testing purposes.

The HTML Test Interface

A simple HTML interface allows you to test an operation. For example, if we click the “emp” operation, the following HTML page is generated:

Test

To test the operation **emp.xquery**, click the 'Invoke' button.

Parameter	Value
id (string) :	<input type="text"/>

To test the operation, simply provide the requested value and click the **Invoke** button. Again, the testing interface is generated dynamically, so the form itself varies based on the operation – if an operation does not require a parameter, it is invoked as soon as you select it. This functionality is supported by the REST technology only.

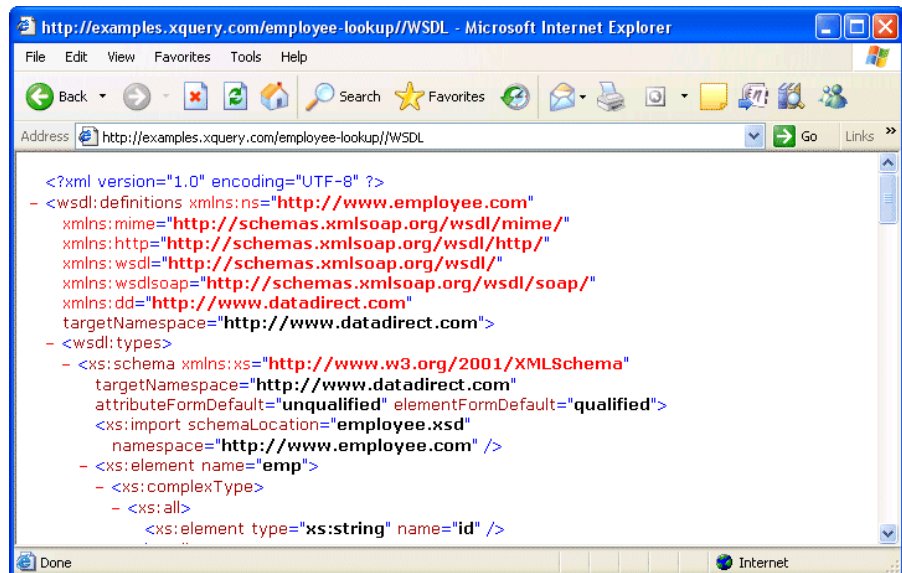
Next Steps

If you choose to use the SOAP transport mechanism in your Web application, you might want to take advantage of the XQueryWebService framework’s ability to generate a WSDL

document. (A WSDL document can be used to create a set of classes that allows you to manipulate a data service as if it was a local library.) Learn about generating a WSDL document in the next section.

Generating WSDL

XQueryWebService automatically generates a Web Service Description Language (WSDL) document based on the XQuery in your Java servlet container's XQuery directory. The WSDL document describes the services that are exposed by a given XQuery. This information can be useful if you plan to provide programmatic access to one or more of those services.



To take a look at the WSDL document generated – in real-time – for the Employee Lookup example, click here:

<http://examples.xquery.com/employee-lookup/WSDL>

If we take a closer look, we see that the WSDL document defines a single service (`<wsdl:service>`), exposed through two ports: SOAP and HTTPGET.

Each query is exposed as a WSDL operation (`<wsdl:operation>`), with each query's external variables exposed as operation parameters. Further, all built-in schema types are preserved in the parameter declaration.

Consider the following external variable declared in our example XQuery:

```
declare variable $id as xs:string external;
```

The following global element appears in the `<wsdl:types>` section of the generated WSDL document:

```
<xs:element name="emp">
  <xs:complexType>
    <xs:all>
      <xs:element name="id" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

The input message references the global element, "emp":

```
<wsdl:message name="empInputMsg">
  <wsdl:part element="dd:emp" name="parameters"/>
</wsdl:message>
```

Currently, XQueryWebService does not support user-defined types for external variables.

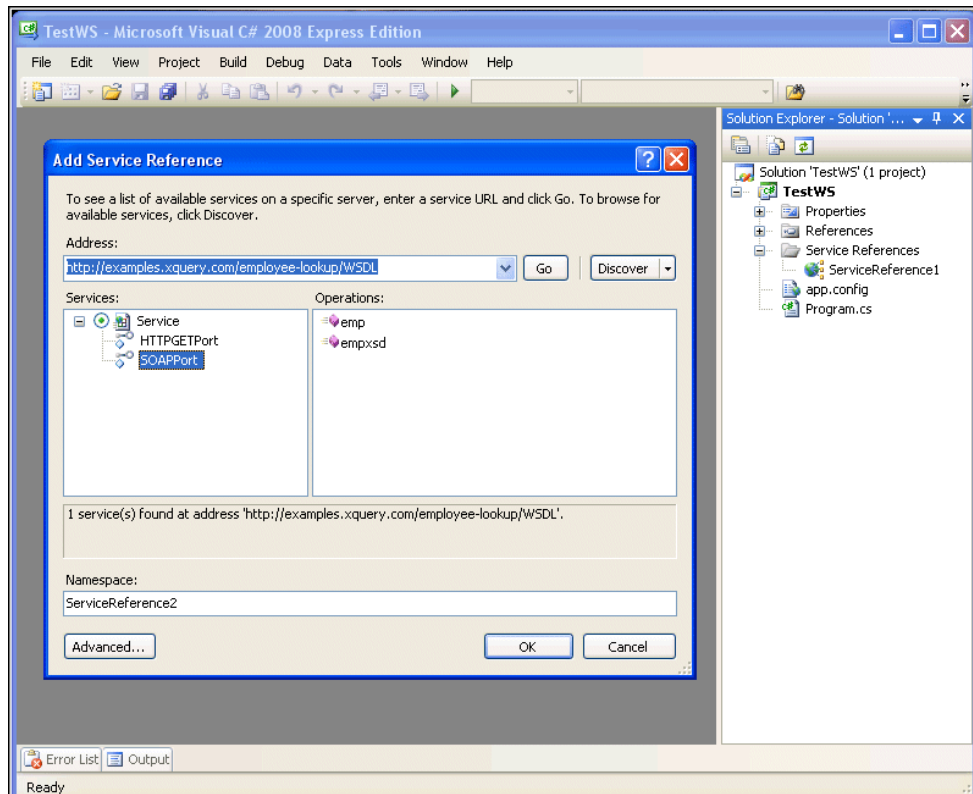
Next Steps

You can use a WSDL document to create a set of classes that can be used to manipulate the data service as if it was a local library. Learn more about using WSDL service references in the next section.

Using WSDL Service References

Modern IDEs like Microsoft Visual Studio and Eclipse provide complete support for consuming Web services – for most of them, making the WSDL document available to the IDE is all that is needed to generate a set of classes that can be used to manipulate the Web service as if it was a local library.

For example, when we open the Employee Lookup WSDL in Microsoft Visual Studio as a Service Reference, the `emp` and `empxsd` operations are exposed, as shown in the following illustration.



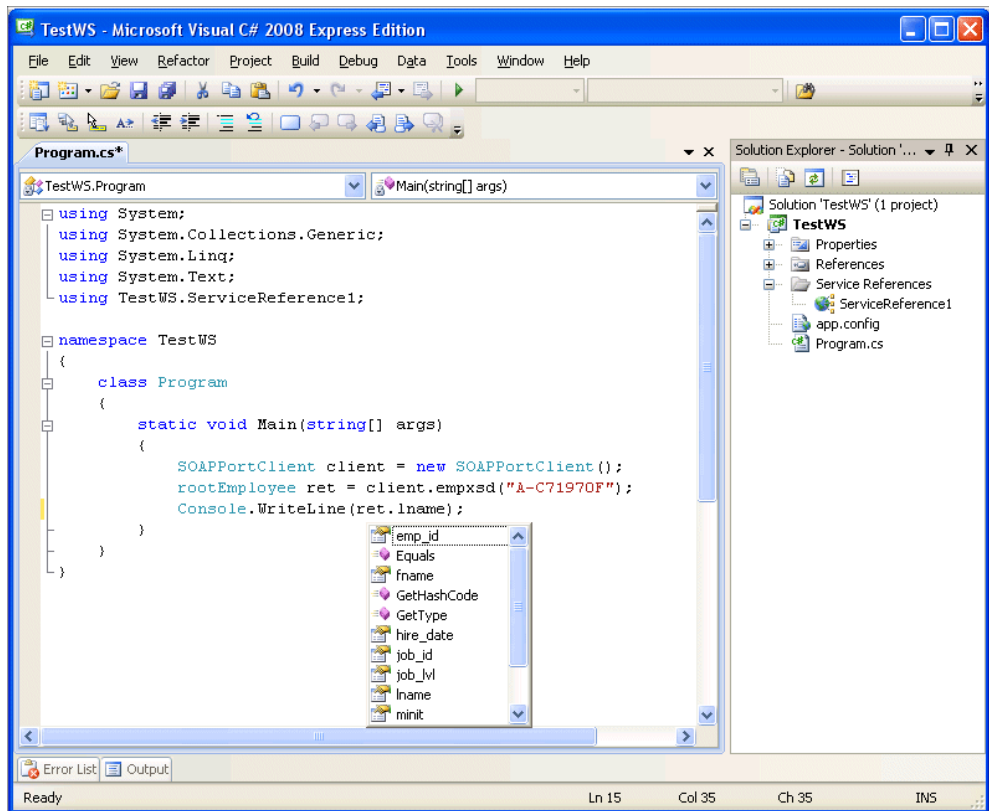
Such a binding framework works extremely well when the WSDL document makes use of XML Schema to describe the SOAP message payloads.

Consider the following simple C# application, which uses the `empxsd` operation, as shown here:

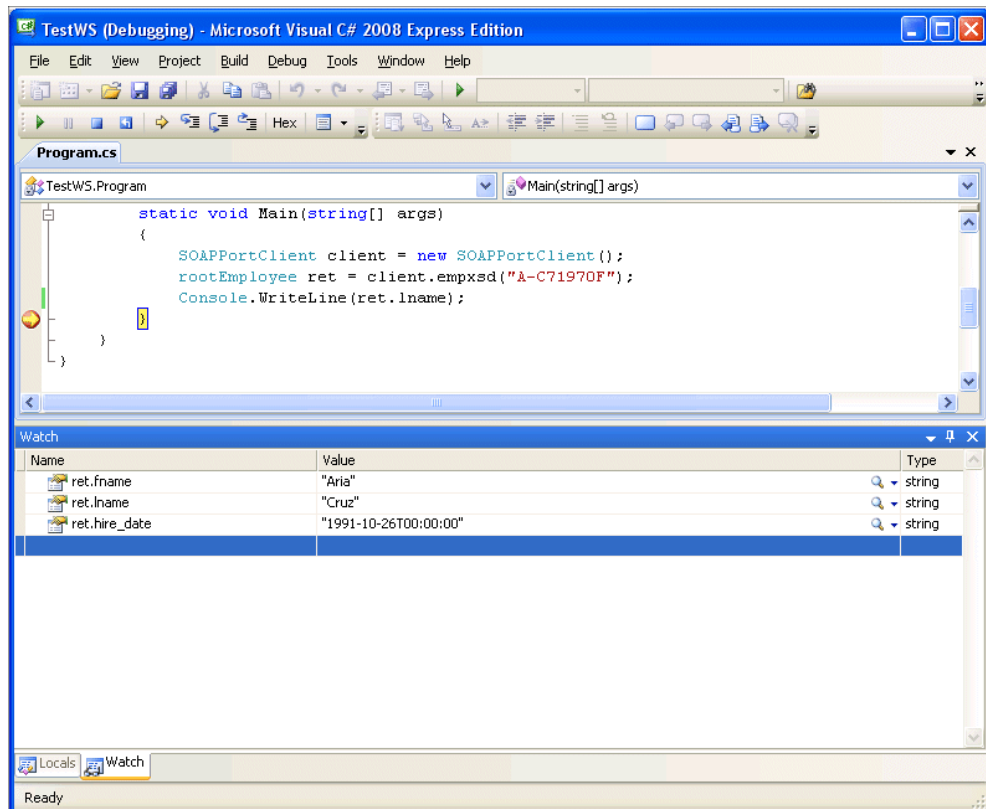
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestWS.ServiceReference1;

namespace TestWS
{
    class Program
    {
        static void Main(string[] args)
        {
            SOAPPortClient client = new SOAPPortClient();
            rootEmployee ret = client.empxsd("A-C71970F");
            Console.WriteLine(ret.lname);
        }
    }
}
```

As seen in the following illustration, schema information about the employee element – employee ID, first and last name, hire date, and so on – is exposed to the IDE, simplifying and enriching the application development process.



If we run this application in debug mode inside Microsoft Visual Studio, we can see that the variables (first name, last name, and so on) are initialized with values from the Web service.



Augmenting WSDL with External XML Schema

To illustrate how XML Schema can be used to augment a data service WSDL, let's revisit the Employee Lookup XQuery (`emp.xquery`), and make a few modifications, as shown:

```

declare variable $id as xs:string external;

<ns:root xmlns:ns="http://www.employee.com">
{
    for $employee in
        collection("pubs.dbo.employee")/employee

```

```

        where $employee/emp_id = $id
        return $employee
    }
</ns:root>

```

This query is almost identical to the one introduced earlier in this example, except that the root element (`<ns:root>`) is now placed in a different namespace. Accordingly, we need to create an XML Schema – we'll call it `employee.xsd` – that describes what the `<ns:root>` element looks like. We'll also put this XML Schema in the same directory as `emp.xquery` and `empxsd.xquery`:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.employee.com"
  xmlns:ns="http://www.employee.com"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="employee" form="unqualified">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="emp_id" form="unqualified" type=
                "xs:NCName"/>
              <xs:element name="fname" form="unqualified" type=
                "xs:NCName"/>
              <xs:element name="minit" form="unqualified"/>
              <xs:element name="lname" form="unqualified" type=
                "xs:NCName"/>
              <xs:element name="job_id" form="unqualified" type=
                "xs:integer"/>
              <xs:element name="job_lvl" form="unqualified" type=
                "xs:integer"/>
              <xs:element name="pub_id" form="unqualified" type=
                "xs:integer"/>
              <xs:element name="hire_date" form="unqualified" type=
                "xs:NMTOKEN"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

If we now open our WSDL URL, we can see that the embedded XML Schema contains an import statement referencing the XML Schema associated with our WSDL:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.datadirect.com"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified">
    <xs:import schemaLocation="employee.xsd"
        namespace="http://www.employee.com"/>
    <xs:element name="empxsd">
        <xs:complexType>
            <xs:all>
                <xs:element name="id" type="xs:string"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="Output" type="xs:anyType"/>
</xs:schema>

```

Now, the SOAP message that describes the operation's return types references the global element `<ns:root>` defined in the WSDL XML Schema:

```

<wsdl:message name="empxsdOutputMsg">
    <wsdl:part element="ns:root" name="parameters"/>
</wsdl:message>

```

I 0 Building a Web Service Client

The previous chapter, [Chapter 9, “Building a Web Service,”](#) described how to use the XQueryWebService framework to expose an XQuery as a Web service on an application server. This chapter describes how to use Stylus XQuery built-in functions to build a Web service client.

This chapter covers the following topics:

- [“Overview” on page 229](#)
- [“HTTP Functions” on page 231](#)
- [“Example: Web Service Client Comparison” on page 244](#)
- [“HTTP Function Request and Response XML Schemas” on page 247](#)

Overview

A *Web service client* is a client-side application that accesses a Web service hosted on a server. Communication between the client machine and host server typically occurs over the Internet using the HTTP protocol. Communication can be direct, or client requests and server responses can be routed through a series of gateways, proxy servers, and tunnels.

Stylus XQuery provides two types of built-in functions to help you write Web service clients – `ddtek:wscall` and HTTP functions.

- `ddtek:wscall` allows you to invoke a Web service operation synchronously using the SOAP protocol over HTTP.

- Stylus XQuery provides full implementation of all HTTP methods like GET, POST, PUT, and DELETE. Stylus XQuery HTTP functions provide a superset of the functionality available with `ddtek:wscall`, allowing more control at a lower level than is possible with `ddtek:wscall`.

Choosing a Function Type

Deciding whether to use `ddtek:wscall` or HTTP functions for your XQuery Web service client depends on a number of factors, the most basic of which is understanding which protocols the Web service supports. For example, if the XQuery Web service supports HTTP, then you must write the XQuery Web service client using HTTP functions. On the other hand, if the XQuery Web service supports SOAP, then you can use either HTTP functions or `ddtek:wscall`.

Other factors you should consider when designing your XQuery Web service client application include:

- Streaming – HTTP functions support streaming of both client-side requests and server-side responses. Streaming is an important consideration for Web service client applications that involve processing large XML documents – when streaming is not used, all data is read into memory before the function is invoked.

See [“Data Streaming” on page 237](#) for more information on this topic.

- Client-server connection – The Stylus XQuery implementation of HTTP functions allows a level of control over the parameters that affect the client-server connection that is not possible using `ddtek:wscall`. Examples of these parameters include specifying the number of retries, socket timeout, and cookie policy management.

See [“HTTP Functions <request> Element” on page 433](#) for more information about specific HTTP parameters.

If the origin server uses REST or plain XML, or if you require direct control over the Web service payload or client-server connection, consider using the Stylus XQuery HTTP functions to build your XQuery Web service client.

HTTP Functions

This section describes the Stylus XQuery HTTP functions you can use to implement an XQuery Web service client, including how to take advantage of some of the low-level connection functionality provided by the HTTP functions that is not available with `ddtek:wscall`.

This section covers the following topics:

- [“Function Overview” on page 231](#)
- [“Connection Authentication” on page 233](#)
- [“Managing Connections and Sockets” on page 235](#)
- [“Data Streaming” on page 237](#)
- [“Response Encoding” on page 238](#)
- [“Managing Cookies” on page 242](#)

Function Overview

Stylus XQuery provides function declarations to support the following HTTP methods on HTTP 1.0 and 1.1:

- DELETE ([“ddtek:http-delete”](#)) – requests that the server delete the specified resource
- GET ([“ddtek:http-get”](#)) – retrieves from the server the resource specified in the URI
- HEAD ([“ddtek:http-head”](#)) – requests that the server return only header information for the specified resource
- OPTIONS ([“ddtek:http-options”](#)) – a request for available

- communication options
- POST (“[ddtek:http-post](#)”) – typically used to submit a resource to the server for additional processing
- PUT (“[ddtek:http-put](#)”) – typically used to insert or replace a server resource
- TRACE (“[ddtek:http-trace](#)”) – requests that the server echo back the request it received

For more information on these functions, including function declaration overloads, see “[Stylus XQuery Built-In Functions](#)” on page 389.

Example – [ddtek:http-get](#)

This example shows a simple use of [ddtek:http-get](#). Here, the XQuery requests the resource `hello.txt` on <http://examples.xquery.com>:

```
ddtek:http-get("http://examples.xquery.com/upload/hello.txt")
```

The result is returned in a `<response>` element, with the data in the `<response-body>`:

```
<response http-version="HTTP/1.1" status-code="200" reason="OK">
  <response-header>
    <header name="ETag" value='W/"5-1243350649093"'/>
    <header name="Date" value="Wed, 24 Jun 2009 20:40:25 GMT"/>
    <header name="Content-Length" value="5"/>
    <header name="Last-Modified" value="Tue, 26 May 2009 15:10:49 GMT"/>
    <header name="Content-Type" value="text/plain"/>
    <header name="Server" value="Apache-Coyote/1.1"/>
  </response-header>
  <response-body>hello</response-body>
</response>
```

For more information about the response header, see “[Response Encoding](#)” on page 238.

Connection Authentication

Before a Web service client application can be run, the client must connect to the origin server. Each connection is authenticated by the Web server. Stylus XQuery supports these authentication methods for HTTP functions:

- Basic (Basic Access Authentication)
- Digest (Digest Access Authentication)
- NTLM (NT LAN Manager)

These authentication methods are separate from any that might be used by applications to access relational database tables using the collection function. See [Chapter 7 “Securing Data Source Connections”](#) to learn about establishing secure connections using NTLM and Kerberos.

The Authentication Process

The type of authentication performed is established by the origin or proxy server. Consider the following example of an invocation of the `ddtek:http-get` function:

```
ddtek:http-get (  
  'http://examples.xquery.com/secure/members/books.xml',  
  <request username="ddtek" password="ddtek"/>  
)
```

Note that the invocation does not specify which authentication method to use, even though the `<request>` element specifies `username=` and `password=` attributes, which suggests an awareness of the need to provide the server with this information. (See [“Specifying HTTP Client-Server Options” on page 244](#) to learn more about the `<request>` element.)

Rather, the required authentication method is provided by the server through a series of challenges to and responses from the XQuery Web service client, as shown in the following simplified exchange:

Stylus XQuery Request

The `ddtek:http-get` function attempts to connect to a resource on `http://examples.xquery.com:`

```
GET http://examples.xquery.com/secure/members/books.xml HTTP/1.1
Host: examples.xquery.com
```

HTTP Server Response

The server denies the request. Included in the response is the authentication method it requires:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="Authentication Test",
qop="auth", nonce="4b3ab6dd951a22816cd32c763ea415e6",
opaque="d14db1a1aba28fd461ce63a8dac78069"
```

Stylus XQuery Request

The request is sent again, this time including the information required by the server – the authorization method, the realm, and a password encrypted using that authentication method:

```
GET http://examples.xquery.com/secure/members/books.xml HTTP/1.1
Authorization: Digest username="ddtek",
realm="Authentication Test",
nonce="4b3ab6dd951a22816cd32c763ea415e6",
uri="/secure/members/books.xml"
```

HTTP Server Response

Once the authorization succeeds, the server responds with an OK and the XQuery can be executed.

```
HTTP/1.1 200 OK
```

This entire exchange, after the initial request by the `ddtek:http-get`, takes place without intervention from the user.

Supported Encryptions

Stylus XQuery supports encryption with the HTTPS (HTTP over SSL) protocol. No additional parameter is required to use SSL (Secure Sockets Layer) – it is indicated by the URI scheme *HTTPS*.

Consider the following example, which connects to the Verisign® secure Web site:

```
declare option ddtek:serialize "indent=yes";
ddtek:http-get("https://www.verisign.com/")
```

This query generates the following result (the response body has been omitted for formatting considerations):

```
<response http-version="HTTP/1.1" status-code="200" reason="OK">
  <response-header>
    <header name="Content-type" value="text/html"/>
    <header name="Date" value="Mon, 01 Jun 2009 15:39:19 GMT"/>
    <header name="Set-Cookie" value="v1st=4A23F627FE4EFAD6; path=/;
      expires=Wed, 19 Feb 2020 14:28:00 GMT; domain=.verisign.com"/>
    <header name="Connection" value="close"/>
    <header name="Server" value="Netscape-Enterprise/4.1"/>
  </response-header>
  <response-body>...</response-body>
</response>
```

Managing Connections and Sockets

Each Stylus XQuery HTTP function is associated with an underlying connection manager. Each connection manager can keep one, and only one, connection open.

When a sequence of calls is directed to the same host, the connection is reused. For example, the following XQuery code executes a sequence of HTTP PUT functions in order to upload a set of XML files from a local directory (c:/docs) to a WebDAV server (<http://examples.xquery.com/upload/>):

```
declare variable $host := "http://examples.xquery.com/upload/";
```

```
<result>{
for $d in collection("file:///c:/docs?select=*.xml")
let $filename := tokenize(document-uri($d), '/') [last()]
return
  ddtek:http-put(concat($host, $filename), $d)
}</result>
```

Contrast the previous example with this one, which uses two connections because it is uploading the same document (books.xml) to two different servers (http://localhost and http://remotehost):

```
<result>{
ddtek:http-put(

  "http://localhost/books.xml", doc("file:///c:/books.xml")),
ddtek:http-put(
  "http://remotehost/books.xml", doc("file:///c:/books.xml"))
}</result>
```

Settings for Connections and Sockets

Stylus XQuery provides numerous settings that allow you more direct control over the connections and sockets associated with your XQuery Web service client. Examples of connection and socket settings you can specify include connection and socket timeout values and the number of retries.

You specify connection and socket settings as attributes of the `<request>` element, which can be included as an additional parameter of the Stylus XQuery HTTP function you are invoking, as shown in the following example:

```
ddtek:http-get (
  "http://examples.xquery.com/upload/dis-logo.psd",
  <request response-data-type="base64" retries="4"/>)
```

Here, the `<request>` element defined as part of the `ddtek:http-get` function specifies that Base64 should be used for

the response encoding, and that the HTTP call should be tried a maximum of four times.

The `<request>` element attributes you can use to manage connection and sockets settings are:

- `connection-timeout`
- `password`
- `protocol-head-body-timeout`
- `protocol-reject-head-body`
- `proxy-host`
- `proxy-password`
- `proxy-port`
- `proxy-username`
- `retries`
- `socket-linger`
- `socket-receivebuffer`
- `socket-sendbuffer`
- `socket-timeout`
- `username`

See [“Specifying HTTP Client-Server Options” on page 244](#) to learn more about the `<request>` element. For a description of these settings as well as a complete list of `<request>` element attributes, see [“HTTP Functions `<request>` Element” on page 433](#).

Data Streaming

All HTTP functions are designed to support *Streaming XML*, a technique that processes data sequentially, allowing efficient transmission of data input and output during query execution.

There might be times, however, when you want to turn off streaming in order to preserve resources on the origin server. When a query invokes a large number of HTTP calls, for example,

each call opens a dedicated connection which can ultimately lead to degraded performance.

Disabling Streaming XML

There are two ways to disable Streaming XML in Stylus XQuery:

- Set `ddtek:xml-streaming="no"` in the query prolog. See [“Using Option Declarations and Extension Expressions” on page 275](#) for more information on this topic.
- Set the `"streaming"` attribute in the `<request>` element to `no`. See [“Specifying HTTP Client-Server Options” on page 244](#) to learn more about the `<request>` element.

Parameter Values and Streaming XML

In contrast to data, which might or might not be streamed, parameter values are always fully materialized before the calling function is invoked. If a query function uses a document as its input, for example, the entire document is loaded in memory before the function is invoked.

See [“Querying Large XML Documents” on page 177](#) for more information on Streaming XML.

Response Encoding

Stylus XQuery automatically encodes the raw data streams returned by `ddtek:http-get` and `ddtek:http-post` functions. The encoded service response is placed in an element referred to as the *payload* for consumption by the XQuery code.

Consider the following simple example. This XQuery:

```
ddtek:http-get("http://examples.xquery.com/upload/hello.txt")
```

returns the following result:

```
<response http-version="HTTP/1.1" status-code="200" reason="OK">
  <response-header>
    <header name="ETag" value='W/"5-1243350649093"' />
    <header name="Date" value="Wed, 24 Jun 2009 20:40:25 GMT" />
    <header name="Content-Length" value="5" />
    <header name="Last-Modified" value="Tue, 26 May 2009 15:10:49 GMT" />
    <header name="Content-Type" value="text/plain" />
    <header name="Server" value="Apache-Coyote/1.1" />
  </response-header>
  <response-body>hello</response-body>
</response>
```

The response-header provides information about the resource being queried (in this case, a file called `hello.txt`), including its Content-Type (which is `"text/plain"`).

The response-body (`"hello"`) represents the data returned by the service, which has been encoded by the Stylus XQuery Web service client.

Encoding Rules

The method used to encode the service response depends on a number of factors, the most important of which is the Content-Type in the response header – if Stylus XQuery recognizes the mime type value in the Content-Type provided in the response header, it provides a suitable encoding method.

Otherwise text encoding is used.

The complete set of encoding rules is summarized here:

- If the origin server does not return a Content-Type in the response header, or if the mime type is not recognized, the response value is encoded as text.

- If the origin server returns a Content-Type in the response header that contains a mime type recognized by Stylus XQuery, one of the following encoding method is used, as appropriate: text, xml, or base64. (See [“Recognized Mime Types” on page 241](#) for a complete list of types and associated encodings.)
- If the Content-Type is a text mime type (text/html, for example) and it includes a charset parameter, as shown in the following example, the charset is used to interpret the bytes in the response:

```
Content-Type: text/html; charset=utf-8
```

If the charset parameter is missing, Stylus XQuery uses ISO-8859-1.

Overriding the Default Encoding

You can override the default encodings used by Stylus XQuery. You might want to do this, for example, when the Content-Type is not specified in the response header but you know what it is.

Consider the following example. Here, we are using the HTTP GET function to query a file:

```
ddtek:http-get (
  "http://examples.xquery.com/upload/dis-logo.psd")
```

The .psd extension indicates that it is a graphics file, which has a mime type of image/x-photoshop. Because this mime type is not recognized by Stylus XQuery, the default encoding method – text – is used. This ultimately causes the query to fail as the text encoding results in the creation of characters like , which are not valid XML.

To avoid this problem, we can override the default encoding using the response-data-type attribute of the <request> element, as shown here:


```
ddtek:http-get (
  "http://examples.xquery.com/upload/dis-logo.psd",
  <request response-data-type="base64"/>)
```

With the response encoding set to a binary type, the query can now be executed successfully:

```
<response http-version="HTTP/1.1" status-code="200" reason="OK">
  <response-header>
    <header name="ETag" value='W/"18016-1243352017406"'/>
    <header name="Date" value="Tue, 23 Jun 2009 21:02:19 GMT"/>
    <header name="Content-Length" value="18016"/>
    <header name="Last-Modified" value="Tue, 26 May 2009 15:33:37 GMT"/>
    <header name="Content-Type" value="image/x-photoshop"/>
    <header name="Server" value="Apache-Coyote/1.1"/>
  </response-header>
  <response-body>OEJQUw...</response-body>
</response>
```

Note that the response body (which has been abbreviated here for formatting considerations) is the base64 encoding of the binary file format.

See [“Specifying HTTP Client-Server Options” on page 244](#) to learn more about the `<request>` element.

Recognized Mime Types

The following table summarizes the mime types recognized by Stylus XQuery and the associated method used encode the response.

Table 10-1. Recognized Mime Types and Associated Encodings

Mime Type	Encoding
application/atom+xml	xml
application/base64	base64
application/mathml+xml	xml

Table 10-1. Recognized Mime Types and Associated Encodings

Mime Type	Encoding
application/rss+xml	xml
application/xhtml+xml	xml
application/xml	xml
application/xslt+xml	xml
application/zip	base64
image/svg+xml	xml
image/bmp	base64
image/gif	base64
image/jpeg	base64
image/png	base64
image/tiff	base64
text/html	text
text/plain	text
text/richtext	text
text/xml	xml
x-gzip	base64
x-compress	base64

Managing Cookies

Cookies are messages exchanged between a Web server and client that are used to manage HTTP state. Cookies commonly store information about visits to a Web site such as a user name, password, the last time a site was visited, and so on. There are several standards for cookies. Some, like [RFC2109](#) and RFC2965, were drafted by the W3C; others, like the Netscape cookie specification, are vendor-specific.

By default, Stylus XQuery supports the [RFC2109](#) standard, but you can use the `cookie-policy` attribute of the `<request>` element to specify how you want your XQuery application to manage cookies. Available cookie management policies are summarized in the following table.

Table 10-2. `<request>` Element `cookie-policy` Parameters

Value	Description
RFC2109	RFC2109 was the first W3C cookies specification. Although widely used, RFC2109 is sometimes too strict for servers supporting other specifications. If you encounter compatibility issues with RFC2109 , consider using RFC2965 .
RFC2965	RFC2965 is the second version of the W3C RFC2109 cookies specification, intended to loosen some of the restrictions that made RFC2109 incompatible with some servers. Some of the key difference of RFC2965 include: <ul style="list-style-type: none"> ■ RFC2965 cookies are port-sensitive ■ Servers that send RFC2965 cookies will use both Set-Cookie2 and Set-Cookie headers; other cookie implementations use only Set-Cookie
netscape	The Netscape cookie specification formed the basis for RFC2109 . However, differences between the two might require the use of Netscape specification on some servers.
ignore_cookies	This setting ignores all cookies – cookies are neither sent nor accepted when the <code>cookie-policy</code> attribute is set to <code>ignore_cookies</code> .

See [“Specifying HTTP Client-Server Options” on page 244](#) to learn more about the `<request>` element.

Specifying HTTP Client-Server Options

Every Stylus XQuery HTTP function allows you to specify client-server options using a `<request>` element, as shown in the following example:

```
ddtek:http-get (  
  "http://examples.xquery.com/upload/dis-logo.psd",  
  <request response-data-type="base64"/>)
```

Here, the `response-data-type` attribute is being used to specify the encoding method to be used for the graphics file `dis-logo.psd`, which is being queried using the `ddtek:http-get` function.

Other examples of `<request>` element attributes include connection and socket timeout values, the number of retries, and HTTP version.

See [“HTTP Functions `<request>` Element” on page 433](#) for a complete list of `<request>` element parameters and their values.

Example: Web Service Client Comparison

This section uses a simple stock quote Web service to illustrate how you might build an XQuery Web service client using both the `ddtek:wscall` and `ddtek:http-*` functions.

Using HTTP Functions

Here is how you might use Stylus XQuery HTTP functions to build an XQuery Web service client application. You start by declaring three variables – \$host, \$payload, and \$options.

The \$host variable is used to specify the endpoint for the Web service – specifically, the URI of the Web Service Description Language (WSDL) that defines the getQuotes operation made available by this Web service:

```
declare variable
```

```
$host := "http://examples.xquery.com/stock-quotes/WSDL";
```

The \$payload variable is used to specify the SOAP message that is submitted to the Web service by the XQuery Web service client. In this example, the SOAP message contains the ticker value (here, it is PRGS).

```
declare variable $payload :=
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <dd:getQuotes xmlns:dd="http://www.datadirect.com">
      <dd:tickers>PRGS</dd:tickers>
    </dd:getQuotes>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>;
```

The \$options variable is used to specify the name of the SOAP action (getQuotes.xquery) that is performed by the SOAP operations (getQuotes) exposed by the XQuery Web service.

```
declare variable $options :=
  <request>
    <request-header>
      <header name="SOAPAction" value="getQuotes.xquery"/>
    </request-header>
  </request>;
```

Once the `$host`, `$payload`, and `$options` variables is defined, you can use them to invoke the `ddtek:http-post` function, as shown here:

```
let $http-response := ddtek:http-post($host, $payload, $options)/response
return
  if($http-response/@status-code eq '200') then
    $http-response/response-body/*
  else $http-response
```

Using ddtek:wscall

Here is the same application written using `ddtek:wscall`:

```
declare variable $host :=
<ddtek:location
  address="http://examples.xquery.com/stock-quotes//WSDL"
  soapaction="getQuotes.xquery"/>;

declare variable $payload :=
<ddtek:getQuotes>
  <ddtek:tickers>PRGS</ddtek:tickers>
</ddtek:getQuotes>;

ddtek:wscall($host, $payload)
```

The declarations of the `$host` and `$payload` variables using the `ddtek:wscall` function are similar to those using `ddtek:http-post`, with a few differences:

- The Web service operation, `getQuotes`, is specified as part of the Web service endpoint in the `$host` declaration, and not as a separate variable (`$options`)
- The `$payload` variable does not need to include the description of the SOAP envelope.
-

HTTP Function Request and Response XML Schemas

This section documents the ddtek-http-request.xsd and ddtek-http-response.xsd XML Schema.

Request XML Schema

If your XQuery Web service client uses a request element (to specify a connection time out or retries, for exmple), it should conform to the ddtek-http-request.xsd XML Schema.

The request header contains all header info that is to be sent to the server. This includes all request headers as specified in the http protocol, as well as server/service specific headers. The parameters are configuration settings. Except for authentication credentials, none of the parameter info provided is transmitted to the server.

If no value is specified for a parameter, the default setting is used.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:schema elementFormDefault="qualified">-
    <xsd:element name="request">-
      <xsd:complexType>-
        <xsd:sequence>-
          <xsd:element name="request-header" minOccurs="0" maxOccurs="1">-
            <xsd:complexType>-
              <xsd:sequence>-
                <xsd:element name="header" minOccurs="0"
                  maxOccurs="unbounded">-
                  <xsd:complexType>
                    <xsd:attribute name="name" type="xsd:string"
                      use="required"/>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</xsd:schema>
```

```

        <xsd:attribute name="value" type="xsd:string"
                      use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="streaming" type="YesNo" use="optional"
               default="yes"/>
<xsd:attribute name="http-version" type="HttpVersion" use="optional"
               default="http_1_1"/>
<xsd:attribute name="connection-timeout" type="xsd:int"
               use="optional"/>
<xsd:attribute name="socket-timeout" type="xsd:int" use="optional"/>
<xsd:attribute name="socket-linger" type="xsd:int" use="optional"/>
<xsd:attribute name="socket-sendbuffer" type="xsd:int"
               use="optional"/>
<xsd:attribute name="socket-receivebuffer" type="xsd:int"
               use="optional"/>
<xsd:attribute name="tcp-nodelay" type="xsd:int" use="optional"/>
<xsd:attribute name="username" type="xsd:string" use="optional"/>
<xsd:attribute name="password" type="xsd:string" use="optional"/>
<xsd:attribute name="proxy-username" type="xsd:string"
               use="optional"/>
<xsd:attribute name="proxy-password" type="xsd:string"
               use="optional"/>
<xsd:attribute name="response-data-type" type="ResponseType"
               use="optional"/>
<xsd:attribute name="cookie-policy" type="CookiePolicy"
               use="optional" default="RFC_2109"/>
<xsd:attribute name="proxy-host" type="xsd:string" use="optional"/>
<xsd:attribute name="proxy-port" type="xsd:int" use="optional"/>
<xsd:attribute name="wrap-exception" type="YesNo" use="optional"/>
<xsd:attribute name="retries" type="xsd:int" use="optional"/>
<xsd:attribute name="protocol-reject-head-body" type="YesNo"
               use="optional"/>
<xsd:attribute name="protocol-head-body-timeout" type="xsd:int"
               use="optional"/>
<xsd:attribute name="serialize" type="xsd:string" use="optional"/>
</xsd:complexType>

```



```

</xsd:element>-
<xsd:simpleType name="YesNo">-
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="yes"/>
    <xsd:enumeration value="no"/>
  </xsd:restriction>
</xsd:simpleType>-
<xsd:simpleType name="HttpVersion">-
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="http_1_0"/>
    <xsd:enumeration value="http_1_1"/>
  </xsd:restriction>
</xsd:simpleType>-
<xsd:simpleType name="ResponseDataType">-
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="xml"/>
    <xsd:enumeration value="text"/>
    <xsd:enumeration value="base64"/>
  </xsd:restriction>
</xsd:simpleType>-
<xsd:simpleType name="CookiePolicy">-
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="netscape"/>
    <xsd:enumeration value="RFC_2109"/>
    <xsd:enumeration value="RFC_2965"/>
    <xsd:enumeration value="ignore_cookies"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
</xsd:schema>

```

Response XML Schema

Response elements returned by Stylus XQuery HTTP functions (like `ddtek:http-put`, for example) conform to the following XML Schema, `ddtek-http-response.xsd`.

The top level attributes from the response element contain the status information returned by the server as outlined in the HTTP

protocol. The attributes include the version information, the status code, and the reason phrase, respectively.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:schema elementFormDefault="qualified">-
    <xsd:element name="response">-
      <xsd:complexType>-
        <xsd:sequence>-
          <xsd:element name="response-header" minOccurs="0" maxOccurs="1">-
            <xsd:complexType>-
              <xsd:sequence>-
                <xsd:element name="header" minOccurs="0"
                  maxOccurs="unbounded">-
                  <xsd:complexType>
                    <xsd:attribute name="name" type="xsd:string"
                      use="required"/>
                    <xsd:attribute name="value" type="xsd:string"
                      use="required"/>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="response-body" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="http-version" type="xsd:string" use="required"/>
  <xsd:attribute name="status-code" type="xsd:string" use="required"/>
  <xsd:attribute name="reason" type="xsd:string" use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</xsd:schema>
```

I I Support for Relational Databases

This chapter explains Stylus XQuery's support for relational databases. It covers the following topics:

- “Querying Relational Data”
- “Querying XML Type Data”
- “Updating Relational Data”
- “Understanding the Transactional Behavior of Stylus XQuery Updates”

See also [Appendix E, “Database Support”](#) for reference information about relational database support in Stylus XQuery, including version support and data-base specific query functions.

Querying Relational Data

In order to query relational data, Stylus XQuery uses the SQL Adaptor to translate XQuery into one or more SQL statements that can be understood – and executed – by the database engine. Where possible, Stylus XQuery leaves intact those XQuery expressions and constructs that are understood by the database engine. In other cases, Stylus XQuery compensates for XQuery expressions and constructs for which there is no direct SQL translation.

Consider the following XQuery, which queries the pub relational database for the title element of the book whose bookid attribute equals 1:

```

<root>
{
  for $book in
    collection("pub.dbo.bookshop")/bookshop/publication/book
  where $book/@bookid = 1
  return $book/title
}
</root>

```

When translated by Stylus XQuery, the result is a single XQuery fragment embedded in the SQL statement, shown here using the Plan Explain feature:

```

sql SQL: SELECT ALL nrm12."COLUMN_VALUE".query('.') AS RACOL1 FROM
"pub"."dbo"."bookshop" nrm4 CROSS APPLY nrm4."publication".nodes('book
[@bookid = 1]/title') nrm12 ("COLUMN_VALUE") ORDER BY nrm4."id" ASC

```

(For more information on Plan Explain, see [“Generating XQuery Execution Plans” on page 307](#).)

Notice that while the syntax between the XQuery expression and the SQL statement differs, the semantics are the same – the FLWOR expression’s `for` clause has been translated as part of the `SELECT FROM` statement; the `where` clause has been translated as the predicate; and so on.

But database engines might not always support XQuery functions one-to-one. Consider the following XQuery – it is similar to the preceding XQuery example in that it is querying the `pub` relational database, but the FLWOR expression `where` clause uses the XQuery contains function:


```

declare variable $title_fragment as xs:string external;
for $book in
collection("pub.dbo.bookshop")/bookshop/publication/book
where contains($book/title, $title_fragment)
return $book

```

While the result is still a single SQL statement, not all of the original XQuery is embedded. If we look at the SQL translation,

we see that the XQuery contains function has been replaced by the SQL LOCATE function:

 **SQL:** SELECT ALL nrm7."COLUMN_VALUE".query('.') AS RACOL1 FROM "pub"."dbo"."bookshop" nrm4 CROSS APPLY nrm4."publication".nodes('book') nrm7 ("COLUMN_VALUE") WHERE ((len(?) + '.') - 1) = 0 OR {fn LOCATE(?, nrm7."COLUMN_VALUE".value('(title)[1]', 'VARCHAR(4000)'))} > 0) ORDER BY nrm4."id" ASC

When XQuery functions are translated to SQL but are not embedded in the XQuery fragment, it is possible that some processing efficiencies, like indexes on the data, are lost. This can happen, for example, when the query results are placed in a transient table for additional processing.

XML and SQL Data Structures

Another difference that must be taken into account when using XQuery to query relational data is structure – the output of a SQL statement is a table (a flat structure), but the typical XML value is a tree. To achieve the required transformation of the result from a flat structure to a tree structure, Stylus XQuery translates the query into two parts: an XML construction part and a SQL part. The XML construction part adds XML tags to the results retrieved from the database to create the hierarchy requested in the query.

Simplifying Generated SQL

The SQL statements that are translated from XQuery can be complex, but Stylus XQuery supports options that create less complex Select statements. These simplified Select statements can improve performance in some cases.

The options discussed in this section affect only XQuery expressions that are executed by the SQL Adaptor, which translates the query into SQL.

String Comparisons and Trailing Spaces

Unlike XQuery string comparison, SQL character comparison is not sensitive to differences in trailing spaces. To accommodate this semantic difference, the SQL statements that Stylus XQuery executes compare both the strings and the length of the strings.

For example, when comparing an Oracle fixed-width character column with a constant value, Stylus XQuery executes a SQL statement that contains:

```
CHARCOL='constant' AND LENGTH(CHARCOL)=LENGTH('constant')
```

The length comparison can be avoided by adding the following option declaration to the query prolog:

```
declare option ddtex:sql-ignore-trailing-spaces "yes";
```

Using this option declaration is also convenient when two fixed-width character columns with different lengths are used in a join condition. For example, assume the following two tables with a fixed-width character column that have different lengths:

```
table1(col char(10))
table2(col char(20))
```

and the following query

```
for $t1 in collection('table1')/table1
for $t2 in collection('table2')/table2
where $t1/col = $t2/col
return
...
```

Even when table1 and table2 contain rows where the col column contains the same value, by default, the values never match

because the lengths are different. Adding ignore-trailing-spaces to the query prolog avoids this possible issue.

String Functions

While many XQuery functions that operate on strings have an equivalent SQL function, XQuery and SQL semantics often differ slightly. The most important differences are how the following string conditions are handled in XQuery functions versus SQL functions:

- Values of empty sequence or empty string arguments
- Trailing spaces

The result of these differences is that the generated SQL for string functions is complex. If both empty sequence and trailing space behaviors are not relevant, complexity can be avoided by adding the following option declaration to the XQuery prolog:

```
declare option ddtek:sql-simple-string-functions "yes";
```

The result of using this option declaration is that Stylus XQuery generates SQL that translates the XQuery function to the equivalent SQL function without taking into account trailing spaces, or empty sequence or empty string arguments.

Following is an overview of the XQuery string functions that are affected by the sql-simple-string-functions option declaration. The following examples assume that the expression is a part of a query that is being executed by the SQL Adaptor. Most of these examples apply to Microsoft SQL Server; however, similar considerations hold true for other databases.

Example: fn:string-length

- Trailing spaces are ignored, for example:

```
fn:string-length("a ") returns 1 instead of 2.
```

- An empty sequence argument is not handled correctly, for example:

`fn:string-length()` returns `()` instead of `0`.

Example: `fn:ends-with`

- Trailing spaces are not handled correctly, for example:

`fn:ends-with('abc ', 'bc ')` returns `false` instead of `true`.

- Empty sequence arguments are not handled correctly, for example:

`fn:ends-with(), ''` returns `false` instead of `true`.

Example: `fn:substring-after`

- Trailing spaces are not handled correctly, for example:

`fn:substring-after('abc def', 'abc ')` returns `def` with a leading space.

- Empty string arguments are not handled correctly, for example:

`fn:substring-after('test', '')` returns `()` instead of `''`.

- Empty sequence arguments are not handled correctly, for example:

`fn:substring-after('test', ())` returns `()` instead of `''`.

Example: `fn:upper-case` and `fn:lower-case`

- An empty sequence argument is not handled correctly, for example:

`fn:upper-case()` returns `()` instead of `''`.

Example: `fn:substring`

- Empty sequence arguments are not handled correctly, for example:

`fn:substring(), 1, 1` returns results in a SQL exception.

- Non positive integer values for start and length argument result in an error.

Example: fn:concat

- Empty sequence arguments are not handled correctly, for example:

`fn:concat('a', (), 'c')` returns `()` instead of `''`.

Example: fn:contains

- If second argument is an empty string, for example:

`fn:contains('ab', '')` returns `false` instead of `true`.

DB2 Decimal to String Cast

The default casting of DB2 decimal values to string values can result in strings with leading 0s, which is not XQuery-compliant. Stylus XQuery generates SQL that removes these 0s. The resulting SQL gets fairly complex. This can be avoided by adding the following option declaration to the XQuery prolog:

```
declare option ddtek:sql-simple-convert-functions "yes";
```

This allows some casts of decimal values to strings to return strings with leading 0s.

Using an Order By Clause

Due to limitations of some SQL databases, it is not always possible to order a SQL result set on an expression that is not part of the Select list. Stylus XQuery supports an option declaration that allows you to choose whether Order By clauses in the generated SQL are explicitly added to the Select list.

The option can be set by adding the following option declaration to the XQuery prolog:

```
declare option ddtex:sql-order-by-on-values "no|yes|noSubquery";
```

- When set to yes, expressions on which to sort are not explicitly added to the Select list of generated SQL Select statements.
- When set to no, values or expressions on which to sort are always added to the Select list; although this typically decreases performance it is required by some databases.
- When set to noSubquery, the behavior is equivalent to yes except when the expression on which to sort is a subquery. In this case, the value noSubquery behaves as if no is specified.

The default value is database dependent:

Database	Default
DB2	no
Informix	noSubquery
MySQL	yes
Oracle	yes
PostgreSQL	yes
Microsoft SQL Server	yes
Sybase	noSubquery

There should be no reason to change the value for Microsoft SQL Server, Oracle, MySQL, or PostgreSQL; however, for DB2, Informix, and Sybase, many queries perform faster when the value is changed to yes.

NOTE: The noSubquery value can optimize the performance of some queries with DB2. When using this value, make sure you set the rewrite-exists-into-count option declaration to inCase (see [“Using a SQL EXISTS Subclause in DB2” on page 259](#)).

Example

For example, assume the XQuery expression below is executed against a Microsoft SQL Server database:

```
for $x in collection('users')/users
order by $x/userid
return $x/name
```

When order-by-on-values is set to no, the SQL statement executed is:

```
SELECT ALL name AS racol1,userid AS racol2
FROM users
WHERE name IS NOT NULL
ORDER BY racol2 ASC
```

When order-by-on-values is set to yes, the SQL statement executed is:

```
SELECT ALL name AS racol1
FROM users
WHERE name IS NOT NULL
ORDER BY userid ASC
```

Using a SQL EXISTS Subclause in DB2

The different DB2 systems impose limitations on the usage of the SQL EXISTS subclause. This option specifies whether to change an EXISTS subclause into a count() > 0 subclause. The option can be set by adding the following option declaration to the XQuery prolog:

```
declare option ddtex:sql-rewrite-exists-into-count "no|yes|inCase";
```

- When set to no, EXISTS subclauses are not rewritten.
- When set to yes (the default for DB2 for z/OS), EXISTS subclauses are always rewritten.
- When set to inCase (the default for DB2 for iSeries), EXISTS subclauses in conditional expressions are rewritten.

Typically, you should not change the default setting, but some XQuery expressions executed against DB2 for z/OS and DB2 for iSeries perform better when this option is set to no. In addition, if order-by-on-values is set to noSubquery for DB2 for Linux/UNIX/Windows, you will get the best performance for the broadest set of queries if you set rewrite-exists-into-count to inCase.

Using BINARY_DOUBLE and BINARY_FLOAT Data Types (Oracle 10g and higher)

By default, Stylus XQuery uses the Oracle NUMBER data type when converting to or constructing XQuery floats or doubles. You can change this behavior for Oracle 10g and higher by adding the following option declaration to the XQuery prolog:

```
declare option ddtex:ora10-use-binary-float-double "yes|no";
```

When this option declaration is set to yes, Stylus XQuery uses the BINARY_FLOAT and BINARY_DOUBLE data types.

Using Stylus XQuery SQL Generation Algorithms

Stylus XQuery uses four SQL generation algorithms, which result in different SQL statements when translating XQuery to SQL. Each algorithm takes a different approach to construct the requested XML hierarchy from the results returned from the SQL statements.

- Merge join
- Nested loop
- Outer join
- Sorted outer union

By default, Stylus XQuery uses the merge join algorithm, which typically gives the best performance. However, when the XML structure is not too deeply nested, the outer join or sorted outer union algorithms give better performance. As a guideline, consider using either outer join or sorted outer union in cases where the XML nesting level is limited to four or less.

See [“Improving Performance” on page 177](#) for additional information about performance.

Merge Join

To construct a correct parent/child relationship, the merge join algorithm creates a first SQL statement that selects all parent values and sorts them on the unique columns of the parents. Then, a second SQL statement joins parent with child values and sorts them again on the unique columns of the parents. The results are processed by moving forward through the results of both SQL statements, linking parent with child node values based on the values of the unique columns.

The merge join algorithm typically gives the best performance. It is the default SQL generation algorithm.

Nested Loop

The nested loop algorithm creates a first SQL statement that selects all values for the parent nodes and a second SQL statement that, for each parent node, selects the values for the associated child nodes.

Use this algorithm when the parent nodes are not uniquely identifiable. In this case, the nested loop algorithm is the only one that returns correct results.

Outer Join

The outer join algorithm creates a single SQL statement that outer joins parent with child node values. The advantage of using this algorithm is that only a single SQL statement is created. The disadvantage is that this single SQL statement can be very complex, as when deeply nested XML structures must be created, for example. In addition, this algorithm requires that for each set of child node values all parent node values are selected as well, which results in redundant information being communicated between the database server and the application.

As a guideline, consider using the outer join algorithm in cases where the XML nesting level is limited to four or less.

Sorted Outer Union

The sorted outer union algorithm creates a single SQL statement that is the union of multiple SQL statements (one for each level in the XML hierarchy). The first SQL statement in the union selects parent node values; the second selects the unique values of the parents joined with the child node values. This approach is recursively applied for each level in the XML hierarchy.

The advantage of using this algorithm is that only a single SQL statement is created. The disadvantage is that this single SQL statement can be very complex and that SQL engines are typically not well-tuned for complex union statements.

As a guideline, consider using the sorted outer union algorithm in cases where the XML nesting level is limited to four or less.

Specifying an Algorithm

You can specify the algorithm to use for a given connection or for an individual query.

To specify an algorithm for a given connection, configure the `JdbcOptions` property of `DDXQDataSource` or the `Options` property of `DDXQJDBCConnection`. For example:

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcOptions("sql-rewrite-algorithm=nested-loop");
```

See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

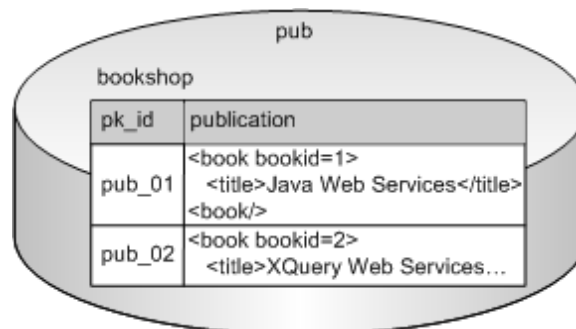
To specify an algorithm for an individual query, add the `sql-rewrite-algorithm` option declaration to the Query prolog. For example:

```
declare option dtek:sql-rewrite-algorithm "outer-join";
```

Querying XML Type Data

DB2, Microsoft SQL Server, and Oracle support a native XML data type, *XML Type*. Stylus XQuery allows you to access XML Type data stored in relational databases using standard XQuery.

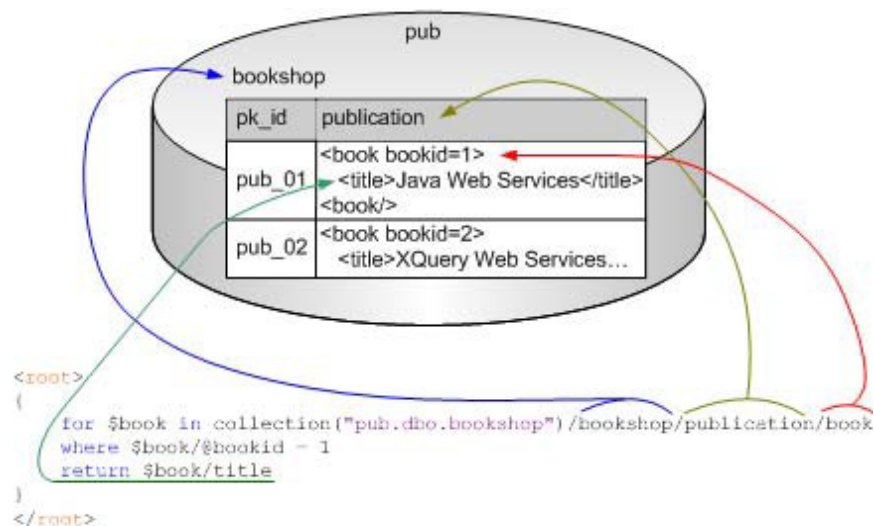
Consider a relational database, `pub`. It contains a table, `bookshop`, with a `publication` column; the `publication` column is defined with the XML Type data type. Data in the `publication` column is stored as XML – there is a `book` element with a `bookid` attribute, and a `title` subelement.



Once you know the structure of the XML in the relational tables you want to query, writing the XQuery is simple. Stylus XQuery's support of XML Type lets you use XQuery to query XML data stored in relational databases. For example, this XQuery returns the title element of the book element whose bookid attribute equals 1:

```
<root>
{
  for $book in
    collection("pub.dbo.bookshop")/bookshop/publication/book
  where $book/@bookid = 1
  return $book/title
}
</root>
```

Let's take a closer look at Stylus XQuery's support of XML Type. As shown in the following illustration, the collection function identifies the resource to be queried – in this case, the table bookshop on the relational database pub. The arrows depict how the XQuery expression following the collection function is used to navigate the XML stored in the table bookshop



The rest of this section provides additional information about using Stylus XQuery to query XML Type data.

Supported Databases

Stylus XQuery is able to directly query database tables with XML Type columns for the following database versions:

- DB2 for Linux, UNIX and Windows v9.1* and higher
- Microsoft SQL Server 2005 and 2008
- Oracle 10gR2 and higher

*XML Type is supported only for these versions of DB2 that use pureXML storage. XML Type is not supported for DB2 databases that use XML Extender.

For more information about how Stylus XQuery translates XQuery into SQL for execution by the database engine, see [“Querying Relational Data” on page 251](#).

Evaluating Queries in Memory

Not all relational databases support XQuery, even if they do support XML Type (versions of Oracle prior to 10gR2, for example). Consider the following example, a simple XQuery that uses the collection function to query a relational table:

```
collection("PurchaseOrder1")/PurchaseOrder1/COLXML/PurchaseOrder[test eq '1']
```

If this XQuery is executed against a relational database that does not support XQuery, Stylus XQuery throws the following error:

```
[DataDirect][XQuery]Error at line 1, column 52. The XMLType column "COLXML" can only be used as return expression or argument of a ddtex-sql function.
```

Similarly, you cannot use XPath expressions, except for the `node()` or `*` steps, or other XQuery expressions on the content of the columns in relational tables. For example consider the database table, `holdingsxml`, which stores XML data in the `userid` and `holdings` columns:

<u>holdingsxml</u>	
<u>userid</u>	<u>holdings</u>
Jonathan	<pre><holdings> <share company="Amazon.com, Inc." userid="Jonathan">3000</share> <share company="eBay Inc." userid="Jonathan">4000</share> <share company="Int'l Business Machines C" userid="Jonathan"> 2500</share> <share company="Progress Software" userid="Jonathan">23</share> </holdings></pre>
Minollo	<pre><holdings> <share company="Amazon.com, Inc." userid="Minollo">3000</share> <share company="eBay Inc." userid="Minollo">4000</share> <share company="Lucent Technologies Inc." userid="Minollo">40000</share> <share company="Progress Software" userid="Minollo">4000000</share> </holdings></pre>

If you specify your XQuery as:

```
collection('holdingsxml')/holdingsxml/holdings//share
```

the following message is returned:

```
Path expressions on XML column "holdings" are not
supported.
```

One way to avoid issues with database support for XQuery is to use the `ddtek:evaluate-in-memory` extension expression.

The `ddtek:evaluate-in-memory` extension expression allows you to evaluate XQuery expressions on the value of the XML column in memory. Consider the following example:

```
let $v1 := collection('holdingsxml')/holdingsxml/holdings
return
  (# ddtek:evaluate-in-memory #)
  {$v1//share}
```

The advantages of using `ddtek:evaluate-in-memory` is that the XQuery is portable across databases, and it allows you to use the complete Stylus XQuery feature set to query the XML stored in the database.

However, using `ddtek:evaluate-in-memory` requires the instantiation of *all* the XML in memory, even if only part of it is needed by the XQuery. The holdings column in the sample database used in the previous example, for example, contained small XML fragments; but there is no practical limit to the size of the XML that can be stored in a relational column, and reading large XML fragments into memory for query processing can decrease performance and affect scalability.

Another potential limitation of using the `ddtek:evaluate-in-memory` extension expression is that it can be used with a only limited set of XQuery expressions.

See [“Using Extension Expressions” on page 285](#) for more information on `ddtek:evaluate-in-memory`.

Using Database-Specific Functions

An alternative to evaluating XQuery in memory is to use database-specific functions to query data stored on relational databases. See [“Database-Specific Query Functions” on page 483](#) to learn how to query XML Type data for databases that do not support XQuery.

Updating Relational Data

Stylus XQuery supports updates to relational database tables from inside an XQuery by providing three built-in functions. These functions are:

- [“ddtek:sql-insert”](#)

- “ddtek:sql-update”
- “ddtek:sql-delete”

The Stylus XQuery relational update functionality is an extension of the XQuery Update Facility (XUF). To learn about Stylus XQuery’s support for XUF and how to use XUF to update XML, see [Chapter 5, “Tutorial: The XQuery Update Facility.”](#)

ddtek:sql-insert

The ddtek:sql-insert built-in function inserts a single record in a database table.

The syntax is:

```
declare updating function ddtek:sql-insert(
  table as xs:string,
  column as xs:string,
  value as item()*,
  ...) external;
```

where:

`table` is the database table in which to insert the record. The semantics of `table` are equivalent to those for `fn:collection`; see [“Specifying Relational Database Tables” on page 118](#).

`column` is the column of the database table in which to insert a value.

`value` is the value to insert into the specified column.

`column` and `value` are a pair in a variable argument list. If `column` is specified without `value`, an error is raised. You can specify multiple values for this pair, as shown in the example.

The following example inserts a new record with three columns into the holdings table. The columns and their values are `userid=Minollo`, `stockticker=TIVO`, and `shares=200`.

```
ddtek:sql-insert("holdings", "userid", "Minollo", "stockticker", "TIVO",
  "shares", 200)
```

Other examples can be found in the [RDBMSUpdate](#) example.

ddtek:sql-update

The `ddtek:sql-update` built-in function updates records in a database table.

The syntax is:

```
declare updating function ddtek:sql-update(
  row as element()*,
  column as xs:string,
  value as item()*,
  ...) external;
```

where:

`row` identifies the records in the database table to update. Each item in the sequence must be a row element of the database table returned by a previous `fn:collection` call.

`column` is the column of the database table to update.

`value` is the new value for the specified column.

`column` and `value` are a pair in a variable argument list. If `column` is specified without `value`, an error is raised.

The following example updates a record in the holdings table – in particular, the record where the `userid` column equals `Minollo` and the `stockticker` column equals `PRGS`. In this record, the `shares` column is updated to 500.

```
ddtek:sql-update (
  collection("holdings")/holdings[userid="Minollo" and stockticker="PRGS"],
  "shares", 500)
```

Other examples can be found in the [RDBMSUpdate](#) example.

ddtek:sql-delete

The ddtek:sql-delete built-in function deletes records in a database table.

The syntax is:

```
declare updating function ddtek:sql-delete(
  row as element(*) external;
```

where:

`row` identifies the records to be deleted. Each item in the sequence must be a row element of the database table returned by a previous `fn:collection` call.

The following example deletes all of the records in the holdings database table where the `userid` column equals Minollo.

```
ddtek:sql-delete(collection("holdings")/holdings[userid = "Minollo"])
```

Other examples can be found in the [RDBMSUpdate](#) example.

Understanding the Transactional Behavior of Stylus XQuery Updates

This section describes how Stylus XQuery supports transactions, transaction isolation levels, and distributed transactions.

Transactions

A *transaction* consists of one or more updating XQueries that have been executed, completed, and then either committed or rolled back.

By default, a Stylus XQuery connection (a new `XQConnection` object) is in auto-commit mode. Auto-commit causes a commit after each XQuery is evaluated.

To disable auto-commit, specify `false` as the argument value for `setAutoCommit`, which is a method of the `XQConnection` interface. For example:

...

```
DDXQDataSource ds = new DDXQDataSource();  
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");  
XQConnection conn = ds.getConnection("myuserid", "mypswd");  
conn.setAutoCommit(false);
```

...

When auto-commit is disabled, the application must either commit or roll back each transaction explicitly. Stylus XQuery, by default, rolls back the active transaction when a connection is closed.

To perform commits and rollbacks, use the `commit` and `rollback` methods, respectively, of `XQConnection`. See [“XQConnection Interface” on page 508](#).

Transaction Isolation Levels

Stylus XQuery supports the following isolation levels as defined in the JDBC interface `java.sql.Connection`:

- `java.sql.Connection.TRANSACTION_READ_UNCOMMITTED` (Read Uncommitted) – Locks are obtained on modifications to the database and held until end of transaction (EOT). Reading from the database does not involve any locking.

- `java.sql.Connection.TRANSACTION_READ_COMMITTED` (Read Committed) – Locks are acquired for reading and modifying the database. Locks are released after reading but locks on modified objects are held until EOT.
- `java.sql.Connection.TRANSACTION_REPEATABLE_READ` (Repeatable Read) – Locks are obtained for reading and modifying the database. Locks on all modified objects are held until EOT. Locks obtained for reading data are held until EOT. Locks on non-modified access structures (such as indexes and hashing structures) are released after reading.
- `java.sql.Connection.TRANSACTION_SERIALIZABLE` (Serializable) – All data read or modified is locked until EOT. All access structures that are modified are locked until EOT. Access structures used by the query are locked until EOT.
- `java.sql.Connection.TRANSACTION_NONE` (None) – Transactions are not supported.

Not all databases support all of these isolation levels, as summarized in following table.

Table 11-1. Isolation Level Support

Database	Read Committed	Read Uncommitted	Repeatable Read	Serializable	None
DB2	X (default)	X	X	X	X
Informix	X (default)	X	X	X	
MySQL Enterprise (InnoDB*)	X (default)	X	X	X	
Oracle	X (default)			X	
PostgreSQL	X (default)			X	
SQL Server	X (default)	X	X	X	
Sybase	X (default)	X	X	X	

* MyISAM and Memory MySQL storage engines are non-transactional.

The names of the DB2 isolation levels do not map one-to-one to the names of the JDBC isolation levels. The following table maps the JDBC isolation levels to the appropriate DB2 isolation levels.

JDBC Isolation Level	DB2 Isolation Level
Read Committed	Cursor Stability
Read UnCommitted	Uncommitted Read
Repeatable Read	Read Stability
Serializable	Repeatable Read
None	No Commit *

* Supported for DB2 for iSeries versions that do not enable journaling.

To set an isolation level for a single connection, specify the appropriate value for the `JdbcTransactionIsolationLevel` property of `DDXQDataSource` (see [Table 6-1 on page 128](#)).

To set an isolation level for multiple connections, specify the appropriate value for the `TransactionIsolationLevel` property of `DXQJDBCConnection` (see [Table 6-2 on page 136](#)).

NOTE: Once a connection is made, the transaction isolation level cannot be changed for that connection (`XQConnection` object).

Distributed Transactions

Stylus XQuery does not support distributed transactions. However, it is possible to have a single Stylus XQuery connection (`XQConnection` object) with multiple underlying JDBC connections and perform updates if the updates target only one of the JDBC data sources. It is also possible during the lifetime of an `XQConnection` object to update two different JDBC data sources, provided this is done in separate transactions and not in a single transaction.

I2 Using Advanced Features

This chapter explains the following Stylus XQuery advanced features and when they are useful:

- [Using Option Declarations and Extension Expressions](#)
- [Querying Multiple Files in a Directory](#)
- [Querying ZIP, JAR, and MS Office Files](#)
- [Using URI Resolvers](#)
- [Analyzing EDI to XML Conversions](#)
- [Generating XQuery Execution Plans](#)
- [Specifying Collations](#)

Using Option Declarations and Extension Expressions

Option declarations provide parameters that modify how Stylus XQuery processes queries. *Extension expressions* are syntactical XQuery constructs that modify how Stylus XQuery processes expressions in a query.

Option Declarations

Option declarations provide parameters that modify how Stylus XQuery processes queries. They are similar to extension expressions, which modify how Stylus XQuery processes expressions in a query.

There are three types of option declarations:

- [Global](#)
- [Relational](#)
- [Database-specific](#)

Global Option Declarations

A global option declaration is used as the default for all XML and relational data sources accessed by XQuery queries in your Java application. Stylus XQuery supports the following global option declarations:

- [automatic-update](#)
- [detect-XPST0005](#)
- [ignore-whitespace](#)
- [plan-explain](#)
- [serialize](#)
- [xml-streaming](#)

Table 12-1. Global Option Declarations

Option Declaration	Description
automatic-update	<p>{yes no}. The <code>ddtek:automatic-update</code> option allows XQuery to automatically update data sources that are accessed through <code>doc</code> and <code>collection</code> functions. Relational data sources cannot be updated using automatic update.</p> <p>Data sources affected by XUF update expressions are physically modified at the end of the XQuery execution. See “Updating Data Sources” on page 112 for more information.</p>

Table 12-1. Global Option Declarations (cont.)

Option Declaration	Description
detect-XPST0005	<p>{yes no}. Determines whether err:XPST0005 is raised during static analysis. If set to yes (the default), the error message err:XPST0005 is raised if the static data type assigned to an XQuery expression other than the expression () or data() is void(). For example, this error is raised if Stylus XQuery can statically determine in a NameTest that a path expression can never match a node, for example, because the name of an element is misspelled or a specified column does not exist in the table.</p> <p>If set to no, err:XPST0005 is not raised during static analysis and the expression is evaluated at runtime.</p>
ignore-whitespace	<p>{yes no}. Gives Stylus XQuery the ability to ignore ignorable whitespace (tabs, linefeeds, carriage returns, and spacebar spaces) when parsing XML documents.</p> <p>If set to no (the default), ignorable whitespace is preserved.</p>
plan-explain	<p>'format=xhtml xml[,resourceLocation=<i>path</i>]'. Determines whether an execution plan is generated for the queries in your application.</p> <p>The value for <i>path</i> must be <i>/install_dir/planExplain</i>, where <i>install_dir</i> is the installation directory of Stylus XQuery (for example, <i>/DDXQ3_0/planExplain</i>).</p> <p>When the Plan Explain feature is enabled, Stylus XQuery generates an execution plan and does not execute the queries. See “Generating XQuery Execution Plans” on page 307 for more information.</p>
serialize	<p>Controls the process of serializing the query results into XML, XHTML, or HTML notation as specified by XQuery 1.0: An XML Query Language, W3C Recommendation 23 January 2007 located at:</p> <p>http://www.w3.org/TR/2007/REC-xquery-20070123/</p> <p>See Appendix D, “Serialization Support” for the serialization parameters that you can set using this option declaration.</p>

Table 12-1. Global Option Declarations *(cont.)*

Option Declaration	Description
xml-streaming	<p>{yes no}. Determines whether Stylus XQuery uses Streaming XML when querying large XML documents. The Streaming XML technique reduces memory consumption. When set to yes, Streaming XML is enabled in the XML Adaptor. The default is yes.</p> <p>See “Querying Large XML Documents” on page 177 for details.</p>

Relational Option Declarations

Relational declarations control the processing of XQuery queries for connections to relational databases; they can be used with any relational database supported by Stylus XQuery.

- [sql-decimal-cast](#)
- [sql-extra-checks-trailing-spaces](#)
- [sql-ignore-trailing-spaces](#)
- [sql-order-by-on-values](#)
- [sql-rewrite-algorithm](#)
- [sql-simple-convert-functions](#)
- [sql-simple-string-functions](#)
- [sql-unicode-literals](#) *(deprecated)*
- [sql-unicode-strings](#)
- [sql-varchar-cast](#)

Table 12-2. Relational Option Declarations

Option Declaration	Description
sql-decimal-cast	<p><i>precision, scale</i>. Determines precision and scale used for <code>xs:decimal()</code>.</p> <p>If you do not specify a precision and scale, the following default values are used:</p> <p>DB2: 30, 15 Informix: 32, 15 MySQL: 64, 30 Oracle: No default Microsoft SQL Server: 38,19 PostgreSQL: No default Sybase: 38, 19</p> <p>You can override the default by using this option declaration.</p>
sql-extra-checks-trailing-spaces	<p>{yes no}. When set to yes, the SQL generated for the XQuery distinct-values functions does not ignore trailing spaces. The default is no.</p>
sql-ignore-trailing-spaces	<p>{yes no}. When set to yes, the generated SQL clauses for string comparison ignore trailing spaces. The default is no.</p>

Table 12-2. Relational Option Declarations (cont.)

Option Declaration	Description
sql-order-by-on-values	<p>{yes no noSubquery}. When set to yes, expressions on which to sort are not explicitly added to the Select list of generated SQL Select statements.</p> <p>When set to no, values or expressions on which to sort are always added to the Select list, which typically decreases performance, but is required by some databases.</p> <p>When set to noSubquery, the behavior is equivalent to yes except when the expression on which to sort is a subquery. In this case, the value noSubquery behaves as if no is specified.</p> <p>The default value is database dependent:</p> <p>DB2: no Informix: noSubquery MySQL: yes Oracle: yes Microsoft SQL Server: yes PostgreSQL: yes Sybase: noSubquery</p> <p>See “Using an Order By Clause” on page 257 for an example.</p>
sql-rewrite-algorithm	<p>{nested-loop merge-join outer-join sorted-outer-union}. Specifies the SQL generation algorithm used by Stylus XQuery when accessing a relational data source. See “Using Stylus XQuery SQL Generation Algorithms” on page 260 for descriptions of these algorithms and information about choosing the appropriate one. The default is merge-join.</p>
sql-simple-convert-functions	<p>{yes no}. When set to yes, the generated convert functions do not consider all semantic differences between XQuery and SQL casts and do, in some cases, return incorrectly formatted string casts of numeric values—for example, leading zeroes in casts of decimal to string with DB2. The default is no.</p>

Table 12-2. Relational Option Declarations (cont.)

Option Declaration	Description
sql-simple-string-functions	{yes no}. When set to yes, the generated SQL string functions do not consider semantic differences between XQuery and SQL functions with regard to empty string or empty sequence (null) arguments. The default is no.
sql-unicode-literals <i>DEPRECATED</i>	This option declaration is recognized for backward compatibility, but we recommend that you use the sql-unicode-strings option declaration.
sql-unicode-strings	<p>{yes no}. Determines whether XQuery literals are translated to SQL literals escaped with the Alternate National Character Set escape character N.</p> <p>If set to yes, XQuery literals are translated to SQL literals escaped with the Alternate National Character Set escape character N. Set the value of this option declaration to yes when a SQL literal contains characters that cannot be translated to the code page of your database. Then, execute the query using this option declaration. This setting is useful when the database table has columns that can contain Unicode information (for example, nvarchar columns).</p> <p>If set to no (the default), XQuery literals are not translated.</p> <p>NOTE: Some databases are significantly slower than others when Unicode comparisons are performed.</p>

Table 12-2. Relational Option Declarations (cont.)

Option Declaration	Description
sql-varchar-cast	<p><i>precision</i>. When multiple XQuery expressions are translated to equivalent SQL statements, a cast to varchar is required. This value defines the varchar precision.</p> <p>If you do not specify a precision, the following default values are used:</p> <p>DB2: 4088 Informix: 254 MySQL: 255 Oracle: 4000 Microsoft SQL Server: 8000 PostgreSQL: No default Sybase: 8000</p> <p>You can override the default by using this option declaration.</p>

Database-Specific Option Declarations

Database-specific option declarations control the processing of XQuery queries for a specific database. Connection-specific option declarations are:

- `sql-ora10-use-binary-float-double`
- `sql-rewrite-exists-into-count`
- `sql-sybase-temptable-index`
- `sql-sybase-use-derived-tables`

Table 12-3. Database-Specific Option Declarations

Option Declaration	Description
<code>sql-ora10-use-binary-float-double</code> Connection-specific	<p>{yes no}. For Oracle 10g and higher. When set to yes, the <code>BINARY_FLOAT</code> and <code>BINARY_DOUBLE</code> data types are converted from XML floats and doubles. The default is no.</p>
<code>sql-rewrite-exists-into-count</code>	<p>{yes no inCase}. For DB2 only. The different DB2 databases impose limitations on the usage of the SQL <code>EXISTS</code> subclause. This option specifies whether to change an <code>EXISTS</code> subclause into a <code>count() > 0</code> subclause.</p> <p>When set to no, <code>EXISTS</code> subclauses are not rewritten.</p> <p>When set to yes (the default for DB2 for z/OS), <code>EXISTS</code> subclauses are always rewritten.</p> <p>When set to inCase (the default for DB2 for iSeries), <code>EXISTS</code> subclauses in conditional expressions are rewritten.</p> <p>Typically, you should not change the default setting, but some XQuery expressions executed against DB2 for z/OS and DB2 for iSeries perform better when this option is set to no. In addition, if <code>sql-order-by-on-values</code> is set to <code>noSubquery</code> for DB2 for Linux/UNIX/Windows, you will get the best performance for the broadest set of queries if you set this option to inCase.</p>

Table 12-3. Database-Specific Option Declarations *(cont.)*

Option Declaration	Description
sql-sybase-temptable-index Connection-specific	{yes no}. For Sybase only. Determines whether an index is created on the temporary table that is used when joining XML and relational data. When set to yes, Stylus XQuery creates this index. Setting this option to yes has a positive impact on performance when hundreds of values are extracted from XML documents and joined with relational data. The default is no.
sql-sybase-use-derived-tables Connection-specific	{yes no}. For Sybase only. Determines whether Stylus XQuery generates SQL statements that use derived tables. Many XQuery expressions require the use of derived tables. However, derived tables are not fully supported in Sybase, and some valid SQL statements that use derived tables fail when executed against a Sybase server. Therefore, Stylus XQuery does not generate SQL statements that use derived tables by default; instead, it creates temporary views, which can adversely affect performance. For some XQuery expressions, the generated SQL that uses derived tables does not fail and returns correct results. To use derived tables, set this option declaration to yes. When using derived tables, performance is improved.

Specifying an Option Declaration

You can specify an option declaration using either of the following methods:

- Using the prolog of the XQuery query. The syntax for specifying an option declaration in a query is:

```
declare option ddtek:name_of_option_declaration "value";
```

For example:

```
declare option dtek:sql-unicode-strings "yes";
```

NOTE: If the option declarations you specify in imported modules conflict with the option declarations you specify in the query or another imported module, Stylus XQuery raises an error.

- Using the properties of the `DDXQDataSource` and `DDXQJDBCConnection` class. To specify an option declaration as global, use the `Options` property of the `DDXQDataSource` class. To specify an option declaration as connection-specific, use the `JdbcOptions` property of the `DDXQDataSource` class or the `Options` property of the `DDXQJDBCConnection` class. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

In the following example, we specify `detect-XPST0005=no` as a global option declaration. It disables the XPST0005 error, which is raised during static analysis under certain conditions (as explained in [Table 12-1](#)). In addition, the example specifies `sql-decimal-cast=20,15` as a connection-specific option declaration. It instructs Stylus XQuery to process a query with a specific precision and scale for `xs:decimal()` values.

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
ds.setOptions("detect-XPST0005=no");
ds.setJdbcOptions("sql-decimal-cast=20,15");
XQConnection conn = ds.getConnection("myuserid","mypswd");

XQExpression xqExpression = conn.createExpression();
FileReader fileReader = new FileReader("xquery_file.xq");
XQSequence xqSequence = xqExpression.executeQuery(fileReader);
```

Using Extension Expressions

Extension expressions provide parameters that modify how Stylus XQuery processes expressions in a query. You can specify extension expressions only in the body of a query. The

only extension expression supported by Stylus XQuery is `evaluate-in-memory` (as defined in [Table 12-4 “Extension Expressions” on page 287](#)).

The syntax for specifying an extension expression in a query is:

```
(# ddtek:name_of_extension_expression #)
```

For example:

```
(# ddtek:evaluate-in-memory #)
```

Suppose a user wants to perform data analysis using the following query, which accesses the historical database table and returns the ratio of the value of `adjustedclose` to the value of `actualclose` for a particular date.

```
for $h in collection('historical')/historical
where $h/ticker = 'AMZN'
return
  <historical>
    {$h/datetraded}
    {$h/adjustedclose div $h/actualclose}
  </historical>
```

Suppose that `actualclose` is 0 for one or multiple dates because of a data entry error. In XQuery, division by 0 raises an error for decimal and integer data types, but not for float and double data types. The user can avoid an error by casting the ratio to a double data type and performing the division in memory by specifying the `evaluate-in-memory` extension expression for the division expression as shown in the following query:

```
for $h in collection('historical')/historical
where $h/ticker = 'AMZN'
return
  <historical>
    {$h/datetraded}
    {(# ddtek:evaluate-in-memory #)
     {xs:double($h/adjustedclose) div $h/actualclose}}
  </historical>
```

[Table 12-4](#) provides a description of the evaluate-in-memory extension expression.

Table 12-4. Extension Expressions

Extension Expressions	Description
evaluate-in-memory	<p data-bbox="654 421 1302 552">Specifies an expression that is evaluated in memory as XQuery, ensuring that it will not be translated to SQL for evaluation in a relational data source. Use this extension expression for the following reasons:</p> <ul data-bbox="654 569 1302 1107" style="list-style-type: none"> <li data-bbox="654 569 1302 916">■ To ensure that Stylus XQuery uses strictly conforming XQuery behavior when processing data from relational data sources. For relational data sources, Stylus XQuery sometimes uses compensation to allow expressions to be evaluated efficiently (as described in “Understanding Compensation” on page 191). When strict conformance to the XQuery specification is more important than efficient data handling, use this extension expression. <li data-bbox="654 933 1302 1107">■ To provide XQuery functionality not typically provided for relational data sources. For example, use this extension expression to perform path expressions on XML stored in the database. <p data-bbox="654 1124 1302 1341">This setting ensures the maximum XQuery conformance, but can significantly degrade performance depending on how it is used. For example, if used in a where clause of a FLWOR expression, it can force all rows of a database table to be evaluated in memory, which degrades performance.</p> <p data-bbox="654 1359 1302 1454">The expression used for evaluation in memory cannot contain the following functions: fn:collection, fn:doc and fn:doc-available.</p>

Querying Multiple Files in a Directory

Stylus XQuery supports the use of `fn:collection` to query multiple XML and non-XML files in a specified directory.

XML Files

In the following example, suppose you have a number of XML files stored in the directory `books`. Each of the files contains information about one book, and you want to create a single XML document that contains a list of all your books.

```
<books>{
for $book in collection("file:///c:/books?select=*.xml")
return
  <myBook>{$book/book/title}</myBook>
}</books>
```

The result would look something like this:

```
<books>
  <myBook>
    <title>Emma</title>
  </myBook>
  <myBook>
    <title>Pride and Prejudice</title>
  </myBook>
  . . .
</books>
```

The function's declaration for this feature is:

```
collection("directory_uri(?option(;option)*)?")
```

where:

directory_uri is a URI referencing a directory. The URI must use the `file://` scheme.


```
option is {(select="REGEX") | recurse={yes | no} |
(sort=[a,t,r]+) | (xquery-regex=(yes|no)) }
```

where:

- `select` contains a regular expression (REGEX), which determines which files in the directory are selected. If `select` is not specified, any file is assumed.
- `sort` determines how the retrieved files are sorted, as follows:
 - `a` sorts alphabetically (ascending).
 - `t` sorts by modification time (beginning with most recent).
 - `r` combined with `a` and `t` reverses the sort order.
- `recurse` determines whether subdirectories are searched. The default is `no`. To search subdirectories, set this option to `yes`, for example:

```
<books>{
for $book in
collection("file:///c:/books?select=*.xml;recurse=yes")
return
  <myBook>{$book/book/title}</myBook>
}</books>
```

- `xquery-regex` determines what type of regular expression syntax is specified in `select`.
 - If set to `no` (the default), the `select` pattern syntax takes the conventional form. For example, `*.xml` selects all files with an `xml` extension. More generally, the `select` pattern is converted to a regular expression by prepending `"^"`, appending `"$"`, replacing `"."` with `"\."`, and replacing `"*"` with `"*"`. Then, the `select` pattern is used to match the file names appearing in the directory using the XQuery regular expression rules. So, for example, you can specify `*(.xml|xhtml)` to match files with either of these two file extensions.

Note however, that special characters used in the URL may need to be escaped using the %HH convention, which can be achieved using the `iri-to-uri` function.

- If set to `yes`, the select pattern syntax as supported by XQuery is assumed. In this case, some characters may need to be escaped such as the backslash character (`\`) in a file name, for example:

```
select=.*\\.xml$ must be select=.*%5C.xml$
```

Non-XML Files

The collection function supports the use of the converter URI, which allows you to use Stylus XML Converters to query non-XML files, such as EDI, binary, and tab- and comma-separated files. For example, this XQuery uses the EDI XML Converter to return a sequence in which each item is an EDI file contained in the directory `C:/myfolder`:

```
fn:collection("converter:///EDI?file:///C:/myfolder")
```

Stylus XQuery also supports additional arguments in `fn:collection` to tune navigation of the specified directory:

```
fn:collection("converter:name:[property_name=value: | property_name=value: | ...]?directory_url(?option(;option)*)?")
```

where:

name is the name of the XML Converter. There are converters for numerous non-XML file types such as EDI, CSV, dBase, and more.

property_name=value are used to specify the properties you want the conversion engine to use when converting a non-XML file to XML. Some properties are shared across converters; others are peculiar to a converter for a given file type.

directory_url and *option* are the same those described in [“XML Files” on page 288](#).

The following examples show how `fn:collection` can be used to query a directory containing EDI files, using the converter URI to specify the EDI to be converted to XML and the properties to be used by the conversion engine.

In this example, X12 elements from all files in the directory `C:\myfolder` are retrieved.

```
fn:collection("converter:///EDI?file:///C:/myfolder")/X12
```

In this example, X12 elements from all files the directory `C:\myfolder` are retrieved, including the ones in sub-folders.

```
fn:collection("converter:///EDI?file:///C:/myfolder?recurse=yes")/X12
```

In this example, X12 elements from all files with extension `.x12` in directory `C:\myfolder` are retrieved, including the ones in sub-folders, and they are sorted in ascending order.

```
fn:collection("converter:///EDI?file:///C:/myfolder?select=*.x12;recurse=yes;sort=a")/X12
```

For More Information

To learn more Stylus XML Converters, the converter URI, and conversion properties, see the *Stylus XML Converters User's Guide and Reference* manual.

<https://www.xmlconverters.com/documentations>

See also “[Collection URI Resolvers](#)” on page 298.

Querying ZIP, JAR, and MS Office Files

Stylus XQuery supports the use of `fn:collection` to directly query XML files archived in a ZIP or JAR file, without first unpacking the archive file. This feature is useful for querying many types of business documents (word-processing documents, spreadsheets, charts, and graphical images such as drawings and presentations) stored in an XML format such as MS Office Open XML and OpenDocument Format.

In the following example, suppose you have multiple XML files archived in the ZIP file `xml.zip`. Each XML file contains information about one book, and you want to create a single XML document that contain lists of all your books.

```
<books>
for $book in collection("zip:file:///c:/xml.zip")//books
return
  <myBook>{$book/book/title}</myBook>
</books>
```

The result would look something like this:

```
<books>
  <myBook>
    <title>Emma</title>
  </myBook>
  <myBook>
    <title>Pride and Prejudice</title>
  </myBook>
  . . .
</books>
```

The function's declaration for this feature is:

```
collection("zip_or_jar_url(?option(;option)*)?")
```

where:

zip_or_jar_url is a URL referencing a ZIP or JAR file. The URL must use the `zip:` or `jar:` scheme.

```
option is {(select="REGEX") | recurse={yes | no} |  
(sort=[a,t,r]+) | (xquery-regex=(yes|no)) }
```

where:

`select` contains a regular expression (REGEX), which determines which files in the directory are selected. If `select` is not specified, any file is assumed.

`sort` determines how the retrieved files are sorted, as follows:

- `a` sorts alphabetically (ascending).
- `t` sorts by modification time (beginning with most recent).
- `r` combined with `a` and `t` reverses the sort order.

`recurse` determines whether subdirectories archived in the ZIP or JAR file are searched. The default is `no`.

To search subdirectories, set `recurse` to `yes`, for example:

```
<books>  
for $book in
```

```
collection("zip:file:///c:/xml.zip?select=*.xml;recurse=yes")//books  
return  
  <myBook>{$book/book/title}</myBook>  
</books>
```

`xquery-regex` determines what type of regular expression syntax is specified in `select`.

- If set to `no` (the default), the `select` pattern syntax takes the conventional form. For example, `*.xml` selects all files with an `xml` extension. More generally, the `select` pattern

is converted to a regular expression by prepending "^",

appending "\$", replacing "." with "\", and replacing "*" with ".*". Then, the select pattern is used to match the file names appearing in the directory using the XQuery regular expression rules. So, for example, you can specify `*(xml|xhtml)` to match files with either of these two file extensions.

Note however, that special characters used in the URL may need to be escaped using the %HH convention, which can be achieved using the `iri-to-uri` function.

- If set to `yes`, the select pattern syntax as supported by XQuery is assumed. In this case, some characters may need to be escaped such as the backslash character (\) in a file name, for example:

```
select=.*\.*.xml$ must be select=.*%5C.*.xml$
```

See also [“Collection URI Resolvers” on page 298](#).

Creating and Updating ZIP Files

You can use the `ddtek:serialize-to-url` function to create new ZIP files and add files to an existing ZIP file. See [“ddtek:serialize-to-url” on page 420](#) for more information.

Using URI Resolvers

This section provides information about URI resolvers for documents, modules, and file collections.

Document URI Resolvers

Stylus XQuery allows an application to specify a custom URI resolver for `fn:doc` and `fn:doc-available`. For example, you may want to create a Java class to resolve custom URLs that point to a proprietary repository that stores your XML documents, like an XML database. This type of custom URI resolver is called a document URI resolver.

The document URI resolver must be a Java class that implements the `javax.xml.transform.URIResolver` interface and the default constructor. The `javax.xml.transform.Source` object returned by the URI resolver must be an instance of one of the following interfaces:

- `javax.xml.transform.stream.StreamSource`
- `javax.xml.transform.sax.SAXSource`
- `javax.xml.transform.dom.DOMSource`
- `javax.xml.transform.stax.StAXSource` (for JVM 1.6 only)*
- `com.ddtek.xquery.StAXSource` (a proprietary Stylus XQuery class for JVMs prior to 1.6)*

* The `StAXSource` must be created with an `XMLStreamReader`; it cannot be created with an `XMLEventReader`.

If you specify a document URI resolver, the rules enforced for URI paths are governed by the syntax specified by your custom URI resolver. See [“XML Data Sources” on page 116](#) for the rules enforced for specifying URIs by the default URI resolver.

You can specify a document URI resolver by configuring the `DocumentUriResolver` property of `DDXQDataSource`. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

Library Module URI Resolvers

Stylus XQuery allows you to customize the mechanism for importing library modules in a query. For example, you may want to create a Java class to resolve custom URLs that point to a repository that stores XQuery modules.

Any custom library module URI resolver must be a Java class that implements the `com.ddtek.xquery.ModuleURIResolver` interface. In addition, it must return an array of Java `StreamSource` objects that identify the contents of a module to be imported.

The interface has one method, `resolve`. You must implement the `resolve` method to resolve the module with the provided module URI, base URI, and location hints, as follows:

```
public StreamSource[] resolve (String moduleURI,
                              String baseURI,
                              String[] locationHints)
                              throws TransformerException
```

The `resolve` method accepts the following parameters specified in the query:

- `moduleURI` is the module namespace URI. This parameter cannot be null.
- `baseURI` is the base URI of the module containing the import module declaration or null if no base URI is known.
- `locationHints` is the array of location hints provided in the `at` clause of the import module declaration. An empty array is specified if no `at` clause is included in the import module declaration.

The `resolve` method returns an array of `StreamSource` objects, each identifying the contents of a module to be imported. Each `StreamSource` must contain a non-null absolute System ID that is used as the location URI of the imported module. Optionally, the `StreamSource` can contain an `InputStream` or `Reader` representing the text of the module. If a representation of the text of the

module is not returned, Stylus XQuery resolves the module using the specified location URI.

If null is returned, Stylus XQuery resolves the module using the default module URI resolver.

You can specify a custom library module URI resolver by configuring the `ModuleUriResolver` property of `DDXQDataSource`. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

Example: Using a Custom Library Module URI Resolver

The following custom library module URI resolver relies on a specific directory to resolve the module. In addition, if a location hint is omitted in the XQuery import statement, Stylus XQuery uses the default module URI resolver.

```
import javax.xml.transform.TransformerException;
import javax.xml.transform.stream.StreamSource;
import java.io.File;
import com.ddtek.xquery.ModuleUriResolver;

public class customModuleResolver implements ModuleUriResolver {
    public StreamSource[] resolve(String moduleURI, String baseURI, String[]
locationHint)throws TransformerException {
        // In this example, the custom behavior is triggered by using
        // a specific moduleURI
        if(moduleURI.equals("http://sharedFunctions.company.com")) {
            File fileSource;
            // If a locationHint is available, we use it; otherwise, we
            // load the default module URI resolver.
            if(locationHint.length > 0)
                fileSource = new File("c:/sharedFunctions/"+locationHint[0]);
            else
                fileSource = new File("c:/sharedFunctions/defaultModule.xquery");
            // More than one StreamSource can be returned;
            // This example only returns one StreamSource.
            StreamSource streamSources[] = {new StreamSource(fileSource)};
            return streamSources;
        }
    }
}
```

```

        return null;
    }
}

```

The following query only specifies the module file name and relies on `customModuleResolver` to import the module from the path `c:/sharedFunctions/module1.xquery`:

```

import module namespace sharedFunctions =
    'http://sharedFunctions.company.com' at 'module1.xquery';
sharedFunctions:func('a','b')

```

The following query omits the location hint. In this case, the query relies on `customModuleResolver` to import a default module (`c:/sharedFunctions/defaultModule.xquery`):

```

import module namespace sharedFunctions =
    'http://sharedFunctions.company.com'; (: no location hint :)
sharedFunctions:func('a','b')

```

In the following query, `customModuleResolver` defaults to the behavior of the built-in Stylus XQuery module URI resolver (note the different namespace URI for the imported module), which tries to locate `module1.xquery` relative to the location of the URL of the query:

```

import module namespace otherFunctions = 'http://other.company.com'
    at 'module1.xquery';
sharedFunctions:func('a','b')

```

Collection URI Resolvers

Stylus XQuery allows an application to specify a custom URI resolver for `fn:collection`. For example, you might want to create a Java class to resolve custom URIs that point to a directory that contains your XML files. This type of custom URI resolver is called a collection URI resolver.

The collection URI resolver must be a Java class that implements the `com.ddtek.xquery.CollectionURIResolver` interface. The URI

resolver returns a `java.util.Iterator` object, which, in turn, must return objects that implement one of the following interfaces:

- `javax.xml.transform.stream.StreamSource`
- `javax.xml.transform.sax.SAXSource`
- `javax.xml.transform.dom.DOMSource`
- `javax.xml.transform.stax.StAXSource` (for JVM 1.6 only)*
- `com.ddtek.xquery.StAXSource` (a proprietary Stylus XQuery class for JVMs prior to 1.6)*

* The `StAXSource` must be created with an `XMLStreamReader`; it cannot be created with an `XMLEventReader`.

Refer to the [Javadoc](#) for details about the `CollectionURIResolver` interface.

You can specify a collection URI resolver by configuring the `CollectionUriResolver` property of `DDXQDataSource`. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

See also [“Querying Multiple Files in a Directory” on page 288](#) and [“Querying ZIP, JAR, and MS Office Files” on page 292](#).

Analyzing EDI to XML Conversions

Stylus XQuery provides several ways to convert non-XML data to XML, including using the converter URI in document and collection functions, as well as built-in functions `ddtek:convert-to-xml`, and `ddtek:analyze-edi-from-*` and `ddtek:edi-to-xml-from-*`.

This section describes how to use Stylus XQuery `ddtek:analyze-edi-from-*` and `ddtek:edi-to-xml-from-*` built-in functions to analyze and convert EDI to XML. It covers the following topics:

- [Overview](#)
- [Built-in EDI Analysis and Conversion Functions](#)
- [Examples](#)

Overview

The following illustration provides an overview of how you can use Stylus XQuery built in functions to analyze and convert EDI to XML. It shows EDI being provided by some entity – an EDI document, EDI data stored on a file system, or EDI provided by a Web service, for example – being passed to the `ddtek:analyze-edi-from-*` and `ddtek:edi-to-xml-from-*` functions. The same EDI stream is specified for each function.

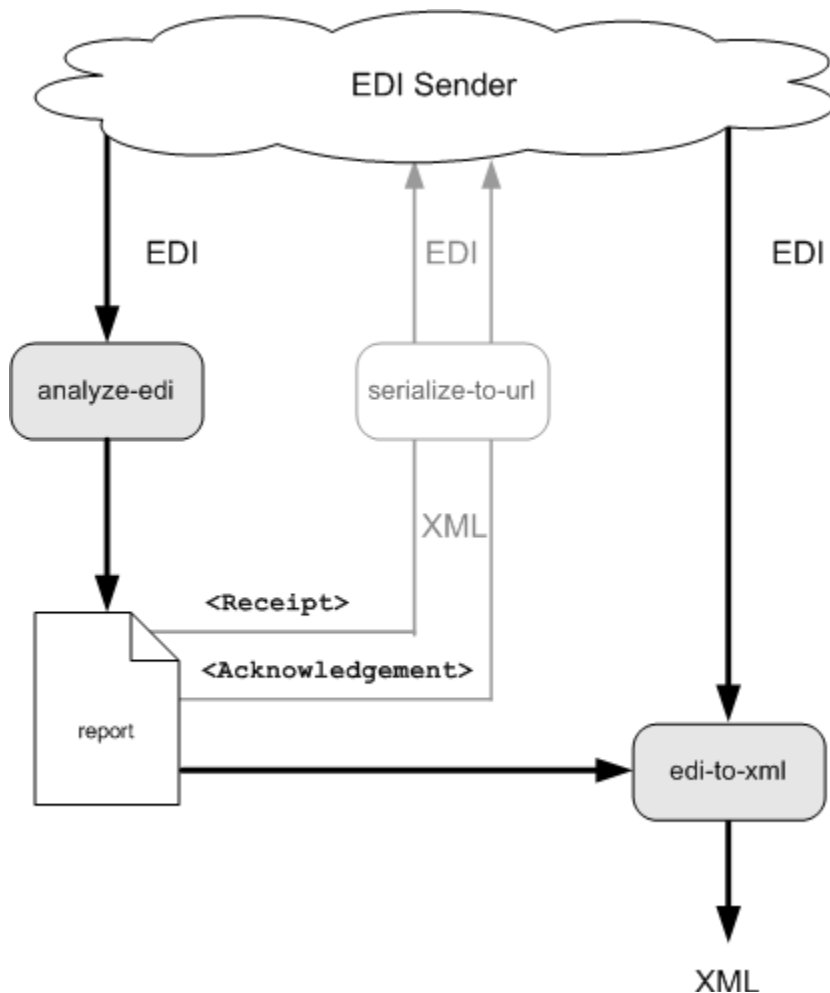
EDI Analysis

First the EDI data stream is analyzed by the `ddtek:analyze-edi-*` function. The `ddtek:analyze-edi-*` function produces an XML report that, among other things, identifies any messages that contain errors. You can write the report to any output you choose – you might want to review the report before converting the EDI to XML, for example – but it is written as a document node by default.

EDI Conversion

Once the analysis is complete, the analysis report is passed to the `ddtek:edi-to-xml-*` function, along with the EDI data stream. The `ddtek:edi-to-xml-*` function uses the errors identified in the

analysis report to filter the EDI, preventing messages containing errors from being converted to XML.



Receipt and Acknowledgement Messages

The analysis report includes a Response element; this element contains Receipt and Acknowledgement subelements. Each subelement holds a complete EDI message in XML format that can be easily manipulated using XQuery and then serialized to

EDI to communicate with the EDI sender whether the transmission was accepted or rejected.

Receipt messages are used to notify the sender that an EDI transmission has been received; acknowledgement messages are used to notify the sender of those messages in the EDI transmission that were rejected because of errors, as well as the nature of the errors. Formats for receipt and acknowledgement messages are dialect-specific.

Supported EDI Dialects

The `ddtek:analyze-edi-from-*` and `ddtek:edi-to-xml-from-*` built-in functions can be used to convert EDI to XML (and not XML to EDI) for the following EDI dialects:

- EDIFACT
- HIPAA
- X12

For More Information

To learn more about analyzing EDI before converting it to XML, including details of the analysis report, see the *Stylus XML Converters User's Guide and Reference* manual.

<https://www.xmlconverters.com/documentation>

Built-in EDI Analysis and Conversion Functions

Stylus XQuery built-in functions for converting EDI to XML:

- Validate an EDI stream and generate an XML report that describes the errors
- Automatically generate accept/reject messages for the EDI sender
- Convert partially valid EDI streams

EDI Analysis Functions

Stylus XQuery `ddtek:analyze-edi-from-*` functions, [ddtek:analyze-edi-from-string](#) and [ddtek:analyze-edi-from-url](#), analyze an EDI stream and generate a report that describes the errors, if any, they detect. These functions can be used in standalone fashion to generate the analysis report, but they are designed to be used with `ddtek:edi-to-xml-from-*` functions as part of the process to convert EDI to XML.

EDI Conversion Functions

Stylus XQuery `ddtek:edi-to-xml-from-*` functions, [ddtek:edi-to-xml-from-string](#) and [ddtek:edi-to-xml-from-url](#), take the report generated by the `ddtek:analyze-edi-from-*` functions and use it to filter detected errors from the EDI before converting it to XML. The `ddtek:edi-to-xml-from-*` functions cannot be used alone to convert EDI to XML – you must pass the report generated by the `ddtek:analyze-edi-from-*` functions.

Specifying the EDI Stream and EDI Conversion Settings

The EDI stream specified in the `ddtek:edi-to-xml-from-*` functions must be the same as that specified in the `ddtek:analyze-edi-from-*` functions for a given XQuery.

For this reason, it is recommended that you use the `ddtek:analyze-edi-from-*` and `ddtek:edi-to-xml-from-*` functions

in pairs – `ddtek:analyze-edi-from-string` and `ddtek:edi-to-xml-from-string`, and `ddtek:analyze-edi-from-url` and `ddtek:edi-to-xml-from-url`, for example.

The same is also true for any conversion properties you specify – any conversion properties specified in the analysis function must also be specified in the conversion function.

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

<https://www.xmlconverters.com/documentation>

Examples

The following examples show how to use Stylus XQuery analyze EDI and convert EDI built-in functions to convert EDI to XML.

EDI Specified as a URL

In this example, the EDI document `code99.x12` is converted to XML:

```
let $edi      := "EDI:tbl=yes?file:///c:/EDI/code99.x12"
let $report   := ddtek:analyze-edi-from-url($edi)
let $ack      := $report/AnalyzeReport/Response/Acknowledgement/X12
let $receipt  := $report/AnalyzeReport/Response/Receipt/X12
let $xml      := ddtek:edi-to-xml-from-url($edi, $report)
return(
  ddtek:serialize-to-url($report, "file:///c:/EDI/code99.x12.report.xml", "")
, ddtek:serialize-to-url($xml, "file:///c:/EDI/code99.x12.xml", "")
, ddtek:serialize-to-url($receipt, "file:///c:/EDI/code99.rec.x12", "method=EDI"
)
, ddtek:serialize-to-url($ack, "file:///c:/EDI/code99.ack.x12", "method=EDI")
)
```

)

In this example, the XQuery produces four files:

- code99.x12.report.xml – The analysis report generated by the [ddtek:analyze-edi-from-url](#) function.
- code99.x12.xml – The XML generated by the [ddtek:edi-to-xml-from-url](#).
- code99.rec.x12 – The receipt message rendered as an XML document.
- code99.ack.x12 – Acknowledgement messages, one for each message in the EDI data stream containing an error, rendered as an XML document.

As noted in [Receipt and Acknowledgement Messages](#), the receipt and acknowledgement messages are dialect-specific. In this example, which uses the X12 EDI dialect, the receipt message conforms to the TA1 message type, and the acknowledge message conforms to the 997 message type. The [ddtek:serialize-to-url](#) function is used to convert code99.rec.x12 and code99.ack.x12 from XML to EDI for transmission back to the EDI sender.

EDI Specified as a String

This example is similar to the first, but this instance the EDI is stored in memory as a string:

```
let $edi := "ISA:00: :00: :01:1515151515 :01:5151515151
:041201:1217:U:00403:000032123:0:P:*~GS:CT:9988776655
:1122334455:20041201:1217:128:X:004030~ST:831:00128001~BGN
:99:88200001:20041201~N9:BT:88200001~TRN:1:88200001~AMT
:2:100000.00~QTY:46:1~SE:7:00128001~GE:1:128~IEA:1:000032123~"
let $url := "EDI:tbl=yes"
let $report := ddtek:analyze-edi-from-string($url, $edi)
let $ack := $report/AnalyzeReport/Response/Acknowledgement/X12
let $receipt := $report/AnalyzeReport/Response/Receipt/X12
let $xml := ddtek:edi-to-xml-from-string($url, $edi, $report)
return (
```

```
ddtek:serialize-to-url($report ,"file:///c:/EDI/code99.x12.report.xml",  
"")  
,ddtek:serialize-to-url($xml ,"file:///c:/EDI/code99.x12.xml", "")  
,ddtek:serialize-to-url($receipt,"file:///c:/EDI/code99.rec.x12","method  
=EDI")  
,ddtek:serialize-to-url($ack ,"file:///c:/EDI/code99.ack.x12",  
"method=EDI")  
)
```

Generating XQuery Execution Plans

The Stylus XQuery feature, Plan Explain, allows you to generate an XQuery execution plan so that you can see how Stylus XQuery will execute your query. For example, if your query accesses a relational data source, the plan will include the SQL statements that Stylus XQuery will send to the database.

The main benefit of using this feature is that you can tune your queries for the best performance possible.

NOTE: The Plan Explain feature is also available in Stylus Studio and Stylus XQuery Editor for Eclipse.

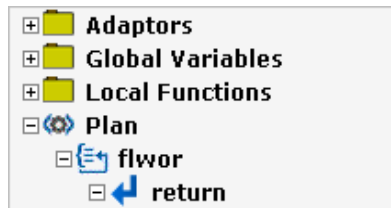
Format of an XQuery Execution Plan

Stylus XQuery outputs the plan in either XHTML format (the default) or XML format. The XHTML format provides a graphical representation of the plan. Stylus XQuery also supports an XML format in the case that you want to create your own graphical representation of the plan or to archive the plan.

XHTML Format

The XHTML representation of an XQuery execution plan is a tree structure that provides the details of how Stylus XQuery will execute the query for which the plan was generated. You can use one of the following browsers to display the XHTML file: Internet Explorer 6.x or 7.x, or Firefox 2.x.

The following figure shows an example of an execution plan in XHTML format.



The tree can contain the following top-level nodes:





- **Adaptors.** This node contains a list of database resources that will be involved in the execution of the query. These resources can include JDBC connections, temporary tables, and deferred SQL statements used in the context of Stylus XQuery update functionality.
- **Global Variables.** This node contains a list of global variables that are available to the query plan, such as external variables defined by the query and variables defined as part of the generation of the execution plan.
- **Local Functions.** This node contains a list of user-defined functions used during the query evaluation. Each user-defined function listed in this node has a plan description associated with it. Plan descriptions are described next.
- **Plan.** This node contains the description of the query execution plan. It contains the nodes of the plan, for example, flwor nodes and the nodes within the flwor nodes such as for, let, and return.

You can navigate the tree to check where variables are defined and where they are referenced. For example, you can navigate from one adaptor's definition to its references and vice-versa. To navigate the tree, you use either the toolbar displayed at the top

of the tree or right-click an item in the tree and use the context-sensitive menu.



The icons on the toolbar perform the following tasks:

Icon	Task
	Go to definition: given a selected variable reference, go to the position in the plan where the variable is defined.
	Go to first reference: given a selected variable definition, go to its first reference in the plan.
	Go to next reference: given a selected variable reference, go to the next reference of the same variable (if any).
	Go to previous reference: given a selected variable reference, go to the previous reference of the same variable (if any).

Enabling Plan Explain

You can enable Plan Explain through XQJ or through the plan-explain option declaration.

Example 1: Enabling Through XQJ

The following code is an example of enabling Plan Explain through XQJ using the proprietary interface `ExtPlanExplain`. This example outputs the query execution plan to an XHTML file named `queryplan.xhtml`.

```
... XQExpression exp = conn.createExpression();
    ExtPlanExplain explain = (ExtPlanExplain)exp;
    XQResultSequence seq = explain.explain("for $item in
        fn:doc('items.xml')/items/item return $item");
    seq.writeSequence(new FileOutputStream("queryplan.xhtml"), null);
    exp.close();
```

Refer to the [Javadoc](#) for details about the `ExtPlanExplain` interface.

The following code is an example of outputting the query execution plan to an XML file named `queryplan.xml`.

```
... XQExpression exp = conn.createExpression();
    ExtPlanExplain explain = (ExtPlanExplain)exp;
    explain.setPlanFormat(PLAN_EXPLAIN_AS_XML);
    XQResultSequence seq = explain.explain("for $item in
        fn:doc('items.xml')/items/item return $item");
    seq.writeSequence(new FileOutputStream("queryplan.xml"), null);
    exp.close();
```


Example 2: Enabling Through an Option Declaration

The following example shows how to enable Plan Explain using the plan-explain option declaration.

```
declare option ddtek:plan-explain "format=xhtml";
for $item in fn:doc('items.xml')/items/item
```

NOTE: When you enable Plan Explain through an option declaration, the query execution plan is returned instead of XQuery results.

See the description of plan-explain in [Table 12-1 “Global Option Declarations” on page 276](#) for more information about this option declaration.

Example of an XQuery Execution Plan

This example XQuery execution plan provides information about how Stylus XQuery translates the following query, which accesses one relational data source, into a SQL Select statement and how XML results are constructed.

```
declare option ddtek:plan-explain 'format=xhtml';
<myHoldings> {
  for $holdings in collection("pubs.dbo.holdings")/holdings
  where $holdings/userid = "Minollo"
  return <holding
    quantity="{ $holdings/shares }">{ $holdings/stockticker/text() }</holding>
}
</myHoldings>
```

In the following execution plan, notice how the Relational Data Source node includes details about the SQL Select statement, as well as information about how the result (\$PT) is constructed.



Specifying Collations

Stylus XQuery allows collations to be specified in the query, for example, in the query prolog or for a specific order by clause. The specified collation, however, is only used for expressions evaluated by the XQuery engine (see [“Stylus XQuery® Architecture” on page 55](#)).

When Stylus XQuery accesses a relational database, it uses the collation used by the database. For consistency, ensure that

the collation used by Stylus XQuery (which can be modified as explained below) is compatible with the collation used by the database. If multiple databases are accessed, ensure that their collations are compatible. For relational sources, the collation used by the database overrides any collation specified in the query.

Stylus XQuery allows you to specify any of the following collations:

- W3C Unicode Codepoint collation. For example:

`http://www.w3.org/2005/04/xpath-functions/collation/codepoint`

- User-defined collation using the following format:

`http://www.datadirect.com/xquery/collation?class=class_name`

where *class_name* is a fully-qualified name of a Java class that implements `java.util.Comparator`. If the collation will be used in functions such as `contains()` and `starts-with()`, this class must also implement `java.text.RuleBasedCollator`.

- Collation using a semicolon-separated list of *keyword=value* pairs in the following format:

`http://www.datadirect.com/xquery/collation?keyword=value[;keyword=value]...`

where the following keywords and values are valid:

lang	Specifies a value that is used to find the collation appropriate to a Java locale. A valid value is any value allowed for the <code>langName</code> or <code>langDef</code> parameters of <code>java.lang.Locale</code> . For example, for US English: en-us
strength	{primary secondary tertiary identical}. Specifies a level of comparison that Stylus XQuery enforces when comparing strings. For example, A/B is a primary difference; a/á is a secondary difference; A/a is a tertiary difference; and a/a is identical. Therefore, if <code>strength=primary</code> , A=a is true; if <code>strength=secondary</code> , A=a is true, but a=á is false; and, if <code>strength=tertiary</code> , A=a is false.

decomposition {none | standard | full}. Determines how the collator handles Unicode characters. Refer to your J2SE documentation for details about how these values affect comparisons of strings.

Stylus XQuery uses the collation URI used by the Java Virtual Machine as the default collation. You can change this value using any of the following methods:

- Specifying the collation parameter in the query prolog
- Specifying the Collation property of DDXQDataSource (see [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#))

Using External Functions

This section explains the types of external functions supported by Stylus XQuery and how to use them. This section covers the following subjects:

- [Supported External Functions](#)
- [Querying Multiple Files in a Directory](#)
- [Using SQL Functions](#)

Supported External Functions

External functions are functions that are implemented outside the query environment. Stylus XQuery supports two types of external functions: Java and SQL.

Java functions might be used to return system information, to invoke a Web service call, or simply to make available a function that is not in the XQuery function library.

SQL functions might be used to invoke a stored procedure or to make available a function that is not in the XQuery function library.

All external functions are namespace qualified, and the namespace that is used tells Stylus XQuery whether the external function is written in Java or in SQL. Before calling an external function, it must be declared in the query.

Example: Java Function (Static Method)

Suppose an application needs to return information about the Java environment in which a query runs. The following Java code defines a class that contains the function to retrieve this information in Java:

```
public class myClass extends Object {
    ...
    public static String myExternalFunction() {
        StringBuffer returnBuffer = new StringBuffer();
        Properties systemProperties = System.getProperties();
        returnBuffer.append("VM Version:"
            + systemProperties.get("java.vm.version") + "\n");
        returnBuffer.append("VM Vendor:"
            + systemProperties.get("java.vm.vendor") + "\n");
        returnBuffer.append("VM Name:"
            + systemProperties.get("java.vm.name"));
        return returnBuffer.toString();
    }
}
```

Now, the function must be declared, as shown in the following query:

```
declare namespace ex="ddtekjava:myClass";
declare function ex:myExternalFunction() as xs:string external;

let $infoString := ex:myExternalFunction()
return
    let($infoString := ex:myExternalFunction()
    return <vm_info>{$infoString}</vm_info>)
```

In the preceding XQuery expression, the XQuery binds a namespace prefix (`ex`) to the fully qualified name of the class that defines the function.

Example: SQL Function

A SQL function can be called in a similar way to calling a Java function. The namespace for a SQL function is the predefined namespace, in this case `http://www.datadirect.com/xquery/sql-function`, which is bound to the predefined prefix `ddtek-sql`. The following example calls the SQL function `rtrim()`.

```
declare function ddtek-sql:reverse($namevalue as xs:string) as
  xs:string external;

for $username in collection('users')//lastname
return
<last_name>{ddtek-sql:reverse($username)}</last_name>
```

Using Java Functions

Stylus XQuery supports Java static methods, Java instance methods, and Java constructors.

The Stylus XQuery type `ddtek:javaObject` allows you to invoke Java methods. The predeclared Stylus XQuery namespace prefix is `ddtek`, and is bound to `http://www.datadirect.com/xquery`. The examples in this section demonstrate the usage of `ddtek:javaObject`.

Declaring Java Functions

Two steps are required to declare Java functions:

- 1 Import the class into the XQuery environment.
- 2 Declare the function.

1. Importing the Class

Before you can declare a Java class, you must import it into the XQuery environment. To do this, you must declare a namespace with a specific URI. The syntax of an import is:

```
declare namespace prefix = "ddtekjava:java class name";
```

where:

prefix is a namespace prefix to associate with the Java class.

java class name contains the complete package and class name of the Java class to import. The specified Java class must be accessed using the Java class loader in the environment in which the query is executed. Typically, this means that the CLASSPATH must contain a reference to the directory or jar file where the Java class can be found. In J2EE server environments, other requirements might exist – the jar or class file might have to be stored in a specific directory, for example, as shown here:

```
declare namespace file = "ddtekjava:java.io.File";
```

If Stylus XQuery cannot find the Java class, it generates a static error.

2. Declaring the Function

Before you can invoke a Java function in a query, you must declare it. How you declare it depends on whether the Java function is a static method or an instance method.

Static Method

To declare a static method, use the following syntax:

```
declare namespace file = "ddtekjava:java.io.File";
declare function namespace:function name(argument list) as return type
  external;
```


For example:

```
declare namespace file = "ddtekjava:java.io.File";
declare function file:createTempFile($prefix as xs:string,$suffix as
  xs:string) as ddtek:javaObject external;
```

See also [“Example: Java Function \(Static Method\)” on page 316](#).

Instance Method

To declare an instance method, follow these steps:

- 1 Import the class to a namespace (see [“1. Importing the Class” on page 318](#)).
- 2 In the cases where you need to explicitly create a new Java object instance, declare a function mapping to a Java constructor unless the Java object type you are declaring has a defined XQuery mapping. A class instance can be an XML type for which a mapping is defined.
- 3 If step 2 is required, declare a function mapping to an instance method declaring ddtek:javaObject as the value for the first parameter. Otherwise, the value of the first parameter is the appropriate XQuery data type.
- 4 Invoke the function using the class instance on which the instance method is invoked as the first argument.

The numbers in the following example correspond to the preceding steps:

```
declare namespace BigInteger = "ddtekjava:java.math.BigInteger"; 1
declare function BigInteger:BigInteger($v as xs:string) 2
  as ddtek:javaObject external;
declare function BigInteger:gcd(
  $this as ddtek:javaObject,
  $val as xs:integer) as xs:integer external; 3

BigInteger:gcd(BigInteger:BigInteger("12"),4) 4
```

The following example does not include step 2 because the Java object to be declared has a defined XQuery mapping.

```
declare namespace BigInteger = "ddtekjava:java.math.BigInteger"; 1
declare function BigInteger:gcd(
  $this as xs:integer,
  $val as xs:integer) as xs:integer external; 3

BigInteger:gcd(12,4) 4
```

Mapping Types Between Java and XQuery for Java External Functions

Before Stylus XQuery passes XQuery arguments to a Java method, it converts the arguments from the XQuery data type to a Java type using the SequenceType specified for the external function when it was declared.

Table 12-5 shows how to map arguments and return types from the Java function declaration to XQuery SequenceTypes to be used in the XQuery function declaration.

Table 12-5. Mapping Types Between Java and XQuery	
Java Type	XQuery SequenceType
boolean	xs:boolean
byte	xs:byte
byte[]	ddtek:javaObject xs:base64binary xs:hexBinary
double	xs:double
float	xs:float
int	xs:int
long	xs:long
short	xs:short
void	empty-sequence() ^a

Table 12-5. Mapping Types Between Java and XQuery *(cont.)*

Java Type	XQuery SequenceType
java.lang.Boolean	ddtek:javaObject xs:boolean[?]
java.lang.Byte	ddtek:javaObject xs:byte[?]
java.lang.Double	ddtek:javaObject xs:double[?]
java.lang.Float	ddtek:javaObject xs:float[?]
java.lang.Integer	ddtek:javaObject xs:int[?]
java.lang.Long	ddtek:javaObject xs:long[?]
java.lang.Object ^b	ddtek:javaObject
java.lang.Short	ddtek:javaObject xs:short[?]
java.lang.String	ddtek:javaObject xs:untypedAtomic[?] xs:string[?]
java.math.BigDecimal	ddtek:javaObject xs:decimal[?]
java.math.BigInteger	ddtek:javaObject xs:integer[?]
java.net.URI	ddtek:javaObject xs:anyURI[?]
java.xml.namespace.QName	ddtek:javaObject xs:QName[?]
javax.xml.transform.Source ^c	ddtek:javaObject document-node()
org.w3c.doc.ProcessingInstruction	ddtek:javaObject processing-instruction() [?]
org.w3c.dom.Attr	ddtek:javaObject attribute() [?]
org.w3c.dom.Comment	ddtek:javaObject comment() [?]

Table 12-5. Mapping Types Between Java and XQuery (cont.)

Java Type	XQuery SequenceType
org.w3c.dom.Document	ddtek:javaObject document-node() [?]
org.w3c.dom.Element	ddtek:javaObject element() [?]
org.w3c.dom.Node	ddtek:javaObject document-node() [?] element() [?] attribute() [?] comment() [?] text() [?] processing-instruction() [?]
org.w3c.dom.Text	ddtek:javaObject text() [?]
boolean, byte, double, float, int, long, short ^d	xs:anyAtomicType? -323
byte[], java.lang.Byte, java.lang.Double, java.lang.Float, java.lang.Integer, java.lang.Long, java.lang.Short, java.lang.String, java.math.BigDecimal, java.math.BigInteger, java.net.URI, java.xml.namespace.QName	xs:anyAtomicType? -323
Any Java data type in this table	item() -323
Any Java object type in this table	item() ? -323
Any Java type in this table	item()* -322 item() + -322
<i>java type</i> [] ^e	ddtek:javaObject <i>XQuery type of this table</i> (* +)

- a. Only for return types.
- b. java.lang.Object or another Java class not listed in this table.

- c. Must be one of the following interfaces:
 `javax.xml.transform.stream.StreamSource`,
 `javax.xml.transform.sax.SAXSource`,
 `javax.xml.transform.dom.DOMSource`,
 `javax.xml.transform.stax.StAXSource`, or
 `com.ddtek.xquery.StAXSource`.

Note that only `javax.xml.transform.dom.DOMSource` is supported for the function's parameters.

- d. Useful to declare Java methods overloaded on argument type. See ["Resolving Function Calls" on page 324](#).
 - e. When the Java parameter type or method return type is an array, either declare the XQuery function to receive or return one of the matching types from this table and add a * or + quantifier. You can also use `ddtek:javaObject` with a * or + quantifier.
-

In cases where [Table 12-5](#) lists multiple mapping options, consider the following information:

- `ddtek:javaObject` cannot be combined with other XQuery types. This means that when the expression you want to pass to a Java function is the result of calling another Java method that returns a `ddtek:javaObject`, you must declare the function as receiving a `ddtek:javaObject` expression. Also, when you declare the return type of a Java function as `ddtek:javaObject`, the returned value can be passed only into another Java method. The returned value cannot be used with any other XQuery expression.
- To map a Java method to an XQuery type when the method is overloaded on parameter type and no common XQuery type can be found for that parameter, you must use a less specific `SequenceType`. Typically, in this case, you would use a less specific `SequenceType` such as `item()`, `xs:anyAtomicType`, or `ddtek:javaObject` depending on the details of the set of overloaded methods. Stylus XQuery attempts to map a given XQuery function invocation to the correct Java function using the static types of the function call arguments instead of the `SequenceType` of the arguments. See the next section [Resolving Function Calls](#) for more information on this topic.

Resolving Function Calls

Before Stylus XQuery attempts to resolve the Java method used to invoke a given XQuery function call, Stylus XQuery identifies the external function declaration.

Then, like any other XQuery function call, the static types of the argument expressions are matched with the `SequenceType` of the function declaration parameters. Static type errors are detected and reported.

Finally, the Java method must be resolved. Sometimes, Stylus XQuery cannot determine how to map an XQuery function declaration and the associated function call to a Java instance method. Ambiguity can occur for the following reasons:

- Java supports method overloading on argument type; however, XQuery does not support function overloading on argument type.
- Invoking Java instance methods from Stylus XQuery requires a first artificial "this" argument. This requirement creates a potential conflict with static methods of the same class that accept a true Java object as their first argument.

Stylus XQuery uses the following steps to determine the Java method to invoke for a given function call. Except for the first step, each of the consecutive steps reduces the number of Java functions that are potential candidates to which to map the XQuery function call.

- 1 Add all public (static and instance) methods and public constructors of the Java class identified through the function's namespace.
- 2 Use a public constructor if the name of the function equals the class name. In the absence of matching public constructors, Stylus XQuery considers Java methods with the same name.

- 3 Retain only Java methods whose names match the name of the function if the name of the function being invoked does not equal the class name.
- 4 Remove all instance methods, unless the first argument in the function call is typed as `ddtek:javaObject` and the Java class associated with the `ddtek:javaObject` equals the class identified by the namespace of the function (see [“Notes About Using Java Instance Methods” on page 326](#)).
- 5 Remove all static methods and constructors whose number of parameters does not match the number of parameters in the XQuery function declaration.
- 6 Remove all instance methods whose number of parameters does not equal the number of parameters of the XQuery function declaration minus one. (This takes into account the first artificial "this" argument that is required when invoking an instance method from XQuery.)
- 7 Remove all Java methods and constructors for which the `SequenceType` of the argument as specified in the function declaration does not match the type of the Java method. The type matching requires a mapping from an XQuery `SequenceType` to a Java type. This mapping is documented in [Table 12-5 on page 320](#). Note that when the function argument is declared as `item()`, `xs:anyAtomicType`, or `ddtek:javaObject`, Stylus XQuery uses the static type of the argument expression of the function call instead of the `SequenceType` of the function declaration. This typically is more accurate and allows correct method resolution in more scenarios.

Unless exactly one method remains after the previously described procedure, one of the following static errors is generated:

Static error during resolving of external Java function.
Ambiguous call to Java external function '<function>'

or

Static error during resolving of external Java function. No matching Java external function found for '<function>'

Note that at execution time:

- Standard Java late binding applies when invoking instance methods.
- If the XQuery external function resolves to an instance method and at runtime the "this" argument evaluates to an empty sequence, the following error is generated:

Runtime error. Value of this pointer is null in call to external Java instance method.

NOTE: Generic methods introduced with J2SE 5.0 are not supported.

Notes About Using Java Instance Methods

- `ddtek:javaObject` can only be used when declaring a `SequenceType` and cannot be used in any other XQuery expression where a `QName` bound to a type is allowed such as `cast`, `treat as`, and so on.
- External variables of type `ddtek:javaObject` are not supported.

- When possible, Stylus XQuery keeps track of the exact Java class that is associated with a given `ddtek:javaObject`-typed expression. This can be useful when trying to map a given function call to a Java method.

When tracking the underlying Java class is not possible, you can use the Stylus XQuery proprietary function, `ddtek:javaCast`, to resolve this issue (see [“ddtek:javaCast” on page 416](#)). Stylus XQuery cannot track the underlying Java class in the following situations:

- The `ddtek:javaObject` expression is the result of executing a recursive function, for example:

```
declare namespace c1 = "ddtekjava:c1";
declare namespace c2 = "ddtekjava:c2";
declare namespace c3 = "ddtekjava:c3";
declare variable $e as xs:integer external;
declare function c1:c1() external;
declare function c2:c2() external;
declare function c3:f($this as ddtek:javaObject)
external;
declare function local:recursive($p as xs:integer)
as ddtek:javaObject {
  if($p le 1) then c1:c1()
  else if($p eq 2) then c2:c2()
  else local:recursive($p - 2)
};
c3:f(local:recursive($e))
```

Assuming that there are two static `c3:f` methods, the first taking an instance of `c1` and the second taking an instance of `c2`, Stylus XQuery is unable to determine statically which one to invoke and generates a static error.

- The static type of an expression is a sequence (or union) of different `ddtek:javaObject` types, for example:

```
declare namespace c1 = "ddtekjava:c1";
declare namespace c2 = "ddtekjava:c2";
declare namespace c3 = "ddtekjava:c3";
```

```

declare function c1:c1() external;
declare function c2:c2() external;
declare function c3:f($this as ddtek:javaObject)
external;
for $x in (c1:c1(),c2:c2()) return c3:f($x)

```

Assuming that there are two static `c3:f` methods, the first taking an instance of `c1` and the second taking an instance of `c2`, Stylus XQuery is unable to determine statically which one to invoke and generates a static error.

- Keeping in mind the preceding information about tracking the extract Java class, it is possible to return a `ddtek:javaObject` value from a query. A Java program containing the query can retrieve the result by:
 - Serializing the result by using one of the following XQSequence methods: `getSequenceAsString`, `writeSequence`, `writeSequenceToSAX`, or `writeSequenceToStream` (see also [Appendix D, “Serialization Support”](#)).
 - Using the XQSequence `getObject` method, which returns the Java object.
 - Using the XQSequence `getLexicalValue` method.

NOTE: Using any other XQJ method to retrieve a `ddtek:javaObject` value generates an error.

See also [“Resolving Function Calls” on page 324](#).

Disabling Java Functions

You can disable the ability to invoke Java functions, for example, for security reasons, by specifying the `AllowJavaFunctions` property of `DDXQDataSource`. See [“DDXQDataSource and DDXQJDBCConnection Properties” on page 128](#).

Using SQL Functions

Stylus XQuery allows you to invoke any SQL function provided by any supported database, including built-in database functions and stored procedures. See [“Example: SQL Function” on page 317](#) for an example of invoking a SQL function.

Requirements and Restrictions

The requirements and restrictions for using SQL functions within a query are:

- You must declare the SQL function as an external function in the <http://www.datadirect.com/xquery/sql-function> namespace, which has a predefined prefix of `ddtek-sql`.

The following example first declares `rtrim()`, and then invokes it within a query:

```
declare function ddtek-sql:rtrim($inp as xs:string) as xs:string external;
for $x in collection('items')//itemno
return
  <a>{ddtek-sql:rtrim(concat($x, ' '))}</a>
```

- You must declare a JDBC Escape function as an external function in the <http://www.datadirect.com/xquery/sql-jdbc-escape-function> namespace, which has a predefined prefix of `ddtek-sql-jdbc`.
- Functions or procedures returning results through output parameters are not supported.
- Functions that return a table are supported if the table function is defined in the source configuration file. This is specified in the `tableFunction` element of the source configuration file. See [“Using SQL Table Functions” on page 337](#) for more information.
- The `SequenceTypes` for parameter and return values in the external function declarations must match the mapping of

database data types to XML schema types as described in the tables listed in [“Data Type Mappings” on page 447](#).

- The SQL function is not supported if the SQL function through an argument or return type refers to a data type for which no mapping is defined. See [“Data Type Mappings” on page 447](#) for the mappings of database data types to XML schema types.
- The quantifier of the return type must be ? or 1.

Microsoft SQL Server Examples

This section presents examples of Microsoft SQL Server functions and describes their equivalents in XQuery.

Example: ddtek-sql:STUFF

The STUFF function inserts the contents of one string in another. The syntax of this function is:

```
STUFF(character_expr,start,length,character_expr)
```

The equivalent XQuery declaration is:

```
declare function ddtek-sql:STUFF(  
  $p1 as xs:string?,  
  $p2 as xs:integer,  
  $p3 as xs:integer,  
  $p4 as xs:string) as xs:string? external;
```

TIP: Although ddtek-sql:STUFF accepts null for its arguments, it is a better idea to provide more precise SequenceType quantifier information, as shown in the preceding XQuery declaration, if the context in which the query is being executed allows such optimization.

An example query using `ddtek-sql:STUFF` is:

```
collection('users')//userid/ddtek-sql:STUFF(.,1,0,
  "userid=")
```

The results are:

```
userid=Jonathan
userid=Minollo
```

Example: `ddtek-sql:STDEV`

The `STDEV` function returns the standard deviation of a set of values. The syntax of this function is:

```
STDEV(expression)
```

where *expression* is a numeric expression.

The equivalent XQuery declaration is:

```
declare function ddtek-sql:STDEV($p as xs:decimal*)
  as xs:double? external;
```

Because `STDEV` is an aggregate function, the quantifier of the `SequenceType` for `$p` must be `*`.

IMPORTANT: You must use `*` as a quantifier for the argument of the `ddtek-sql:STDEV` declaration. If you do not, a SQL error is raised.

An example of invoking `ddtek-sql:STDEV` is:

```
declare function ddtek-sql:STDEV($p as xs:decimal*)
  as xs:double? external;
ddtek-sql:STDEV(collection('historical')//volume)
```

Assume that another invocation of `ddtek-sql:STDEV` within the same query needs to operate on `xs:double` arguments. In this situation, you cannot declare the argument of `ddtek-sql:STDEV` as `xs:decimal*`. Instead, declare the argument as `xs:anyAtomicType*`, as shown:

```
declare function ddtek-sql:STDEV($p as xs:anyAtomicType*)
```

```

    as xs:double? external;
ddtek-sql:STDEV(
    collection(' historical '//volume/xs:decimal(.)),
ddtek-sql:STDEV(
    collection('historical')//actualclose/xs:double(.))

```

DB2 Examples

This section presents examples of DB2 functions and describes their equivalents in XQuery.

Example: ddtek-sql:encrypt

The encrypt function returns an encrypted value. The syntax of this function is:

```

encrypt(StringDataToEncrypt, PasswordOrPhrase,
       PasswordHint)

```

The equivalent XQuery declaration is:

```

declare function ddtek-sql:encrypt($data as xs:string,
                                   $password as xs:string,
                                   $hint as xs:string)
    as xs:string external;

```

An example query using ddtek-sql:encrypt is:

```

declare function ddtek-sql:encrypt($data as xs:string,
                                   $password as xs:string,
                                   $hint as xs:string)
    as xs:string external;

for $x in collection('users')/users
return
<user id='{ $x/userid }' encrypted='{ ddtek-sql:encrypt
    concat($x/firstname,$x/lastname,$x/othername),
    'secret','hint') }' />

```

This example returns:

```
<user
```

```

    id="Jonathan"
    encrypted=
"089B6504E404BFD568696E743A5B64F5A7838CEEEE66DE7F9C5CD92D5E70954C00A81E71"/>
<user
    id="Minollo"
    encrypted="08C04004E404A5D568696E742C93D28C8A2946BF74DB19F6CA6B27BD"/>

```

Example: ddtek-sql:variance

The variance function returns the variance of a set of numbers.
The syntax of this function is:

```
variance(numeric-expression)
```

The equivalent XQuery declaration is:

```
declare function ddtek-sql:variance($inp as xs:decimal*) as
    xs:double external;
```

An example query using ddtek-sql:variance is:

```

declare function ddtek-sql:variance($inp as xs:decimal*) as
    xs:double external;

for $x in
distinct-values(collection('historical')/historical/ticker)
return
<ticker-variance
    ticker='{ $x }'
    variance='{ ddtek-sql:variance(
        collection('historical')/historical[ticker eq
            $x]/adjustedclose) }' />

```

Because variance is an aggregate function, the quantifier of the SequenceType for \$inp must be *.

The example returns:

```

<ticker-variance ticker="AAPL" variance="136.3814396049211"/>
<ticker-variance ticker="ADBE" variance="302.2495900777491"/>
<ticker-variance ticker="AMZN" variance="559.4292663498876"/>
...

```

Oracle Examples

This section presents examples of Oracle functions and describes their equivalents in XQuery.

Example: ddtek-sql:DECODE

The DECODE function behaves as a SQL IF-THEN-ELSE statement. The syntax of this function is:

```
DECODE(expr, search, result
        [, search, result ]...
        [, default ]
      )
```

This function is overloaded; therefore, it must be declared multiple times, for example:

```
declare function ddtek-sql:DECODE(
  $p1 as xs:string,
  $p2 as xs:string,
  $p3 as xs:string,
  $p4 as xs:string) as xs:string external;
declare function ddtek-sql:DECODE(
  $p1 as xs:string,
  $p2 as xs:string,
  $p3 as xs:string,
  $p4 as xs:string,
  $p5 as xs:string,
  $p6 as xs:string) as xs:string external;

for $h in collection('holdings')/holdings
let $ticker := $h/stockticker
let $description := ddtek-sql:DECODE(
  $ticker,
  'PRGS','Progress Software Cooperation',
  fn:concat('Sorry but ', $ticker, ' is not a recognized
    ticker'))
let $holder := ddtek-sql:DECODE(
  $h/userid,
  'Jonathan', 'Mr John',
  'Minollo', 'Senior Minollo',
```



```

    '????')
return
    <who-owns-what
        description='{$description}'
        holder='{$holder}' />

```

This example returns:

```

<who-owns-what description="Progress Software Cooperation" holder="Mr John"/>
<who-owns-what description="Progress Software Cooperation"
  holder="Senior Minollo"/>
<who-owns-what description="Sorry but AMZN is not a recognized ticker"
  holder="Mr John"/>
<who-owns-what description="Sorry but AMZN is not a recognized ticker"
  holder="Senior Minollo"/>
...

```

Using User-Defined Functions

User-defined functions are available for all supported databases. This section provides examples for DB2, Microsoft SQL Server, and Oracle.

For the following examples, assume a user-defined function named `FUNC_TAN`.

Example: DB2

```

CREATE FUNCTION FUNC_TAN (X DOUBLE) RETURNS DOUBLE
LANGUAGE SQL CONTAINS SQL NO EXTERNAL ACTION DETERMINISTIC
RETURN SIN(X)/COS(X)

```

Example: Microsoft SQL Server

```

CREATE FUNCTION FUNC_TAN (@X float) RETURNS float
BEGIN RETURN (SIN(@X)/COS(@X)) END

```

Example: Oracle

```
CREATE OR REPLACE FUNCTION FUNC_TAN (X IN FLOAT) RETURN
  FLOAT
IS TMP FLOAT;
BEGIN
  TMP := SIN(X)/COS(X);
  RETURN (TMP);
END;
```

The following XQuery expression declares and invokes the function:

```
declare function ddtek-sql:FUNC_TAN($x as xs:double?)
  as xs:double? external;
for $v in collection('some_table')//some_numeric_column
return ddtek-sql:FUNC_TAN($v)
```

Using JDBC Scalar Functions

In addition to allowing you to invoke any SQL function provided by any supported database, including built-in database functions, Stylus XQuery allows you to use JDBC scalar functions, which are not database specific. Refer to the JDBC specification for a complete list of JDBC scalar functions.

You must declare a JDBC scalar function in the <http://www.datadirect.com/xquery/sql-jdbc-escape-function> namespace, which has a predefined prefix of `ddtek-sql-jdbc`.

For example:

```
declare function ddtek-sql-jdbc:user() as xs:string external;
<holding-report generated-by='{ddtek-sql-jdbc:user()}'>{
  for $h in collection('holdings')/holdings
  return
    <holding-info userid="{ $h/userid}" ticker="{ $h/stockticker}" />
}</holding-report>
```

Using SQL Table Functions

Stylus XQuery supports table functions, including user-defined table functions, for DB2, Informix, Oracle, PostgreSQL, and Microsoft SQL Server.

To use SQL table functions, you must declare the function and the structure of the returned table using the `tableFunction` element in the source configuration file. See [Appendix H “Source Configuration File” on page 533](#) for details.

DB2 has a system table function, `SYSPROC.DB_PARTITIONS`, that returns system information of the DB2 instance. (Refer to your IBM DB2 documentation for details about this table function.) To invoke this function, configure the source configuration file as shown:

```
<schema name="SYSPROC">
...
  <tableFunction name="DB_PARTITIONS">
    <resultSet>
      <column name="PARTITION_NUMBER" schemaType="short"/>
      <column name="HOST_NAME" schemaType="string"/>
      <column name="PORT_NUMBER" schemaType="short"/>
      <column name="SWITCH_NAME" schemaType="string"/>
    </resultSet>
  </tableFunction>
...
</schema>
```

Once you have configured the source configuration file, declare the function in the `ddtek-sql` namespace (<http://www.datadirect.com/xquery/sql-function>) and use a return type of `document-node(element())`. For example:

```
declare function ddtek-sql:XVS.SYSPROC.DB_PARTITIONS() as
  document-node(element()) external;
ddtek-sql:XVS.SYSPROC.DB_PARTITIONS()
```

One possible result for this example is:

```
<DB_PARTITIONS>
  <PARTITION_NUMBER>0</PARTITION_NUMBER>
  <HOST_NAME>the_host</HOST_NAME>
  <PORT_NUMBER>0</PORT_NUMBER>
  <SWITCH_NAME>the_switch_name</SWITCH_NAME>
</DB_PARTITIONS>
```

A second example:

```
declare function ddtek-sql:XVS.SYSPROC.DB_PARTITIONS() as
  document-node(element()) external;
for $x in ddtek-sql:XVS.SYSPROC.DB_PARTITIONS()/DB_PARTITIONS
return
  <partition number='{ $x/PARTITION_NUMBER}' host='{ $x/HOST_NAME}' />
```

One possible result for this example is:

```
<partition number="0" host="the_host"/>
```

A XQuery Support

This appendix describes how Stylus XQuery supports XQuery expressions, functions, and operators according to the following specifications:

- XQuery 1.0: An XML Query Language, W3C Recommendation 23 January 2007 located at:
<http://www.w3.org/TR/2007/REC-xquery-20070123/>
- XQuery 1.0 and XPath 2.0 Data Model (XDM), W3C Recommendation 23 January 2007 located at:
<http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/>
- XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation 23 January 2007 located at:
<http://www.w3.org/TR/2007/REC-xpath-functions-20070123/>
- XQuery 1.0 and XPath 2.0 Formal Semantics, W3C Recommendation 23 January 2007 located at:
<http://www.w3.org/TR/2007/REC-xquery-semantics-20070123/>
- XQuery 1.1 W3C Working Draft 3 December 2008 located at:
<http://www.w3.org/TR/xquery-11/>

Terminology

The tables in this appendix that present XQuery support information use the following terms to describe this support for both XML and relational sources.

Term	Definition
Supported	Stylus XQuery supports the feature, function, or operator with no exceptions.

Term	Definition
Supported with comment	Stylus XQuery supports the feature, function, or operator with the comment noted.
Not supported	Stylus XQuery does not support the feature, function, or operator and raises an error.

The numbers in the main section in this appendix correspond to the sections in the XQuery 1.0 [W3C Recommendation 23 January 2007](#) where that topic is discussed.

Except where noted, all features are supported for both XQuery 1.0 and XQuery 1.1. Features that are supported *only* in XQuery 1.1 are indicated with a symbol like this: 1.1

In This Appendix

This appendix contains the following sections:

- “2 Basics” on page 341
- “3 Expressions” on page 347
- “4 Modules and Prologs” on page 357
- “5 Conformance” on page 358
- “Namespaces” on page 360

2 Basics

This section describes how Stylus XQuery supports the following XQuery basics.

- [“Expression Context” on page 342](#)
- [“Processing Model” on page 345](#)
- [“Error Handling” on page 345](#)
- [“Concepts” on page 346](#)
- [“Types” on page 346](#)
- [“Comments” on page 347](#)

Expression Context

Table A-1 describes the XQuery expression context for Stylus XQuery.

Table A-1. XQuery Expression Context

XQuery Language	Support
Static context	
XPath 1.0 compatibility mode	This setting is always false as required by the XQuery specification.
Statically known namespaces	Supported. See “Namespaces” on page 360 for details on predefined namespaces.
Default element/type namespace	The default value is no namespace. This value can be set in the query prolog.
Default function namespace	The default value is http://www.w3.org/2005/xpath-functions . This value can be set in the query prolog.
In-scope schema definitions	Only predeclared types are supported. See “Supported XQuery Atomic Types” on page 459 for a list of supported types.
In-scope variables	This is determined by the XQuery expression.
Context item static type	Supported except for initial context item.
Function signatures	See Appendix B “Functions and Operators” on page 361 for a list of supported XQuery functions. User-defined functions are also supported.
Statically known collations	The collations supported by the Java Virtual Machine.
Default collation	<p>Either the Java Virtual Machine collation or the one specified in the source configuration file.</p> <p>For relational sources, the default value is the collation used by the database. To ensure consistency, make sure that the collation used by the Java Virtual Machine is compatible with the collation used by the database. If multiple databases are used, make sure that their collations are compatible.</p> <p>See “Specifying Collations” on page 312 for instructions on specifying collations.</p>

Table A-1. XQuery Expression Context (*cont.*)

XQuery Language	Support
Construction mode	Supported with comment: preserve is supported. strip is not supported.
Ordering mode	Supported.
Default order for empty sequences	The following algorithm is used: <ol style="list-style-type: none"> 1 The default is empty least. 2 The default is overwritten by the default ordering behavior of the first JDBC connection defined using XQJ or the source configuration file. The default ordering of the first JDBC connection is defined by the database: <ul style="list-style-type: none"> DB2: empty greatest Informix: empty least Microsoft SQL Server: empty least Oracle: empty greatest PostgreSQL: empty greatest Sybase: empty least
Boundary-space policy	The default value is strip (white space is removed during processing). This value may be set in the query prolog.
Copy-namespaces mode	inherit,preserve mode is supported.
Base URI	Supported. You can set this value in the query prolog or in the DDXQDataSource class by configuring the BaseUri property.
Statically known documents	Not supported.
Statically known collections	Data types of the data stored in database tables that can be referenced with fn:collection() are statically known although these data types cannot be accessed from within the expression. See “Data Type Mappings” on page 447 for a list of supported data types.
Statically known default collection type	Not supported.

Table A-1. XQuery Expression Context (cont.)

XQuery Language	Support
Dynamic context	
Context item	Supported, except for initial context item.
Context position	Supported. Context-position dependent expressions are compensated for relational sources.
Context size	Supported.
Variable values	Supported.
Function implementations	Supported.
Current dateTime	Supported. The current date and time are obtained from the Java Virtual Machine; however, queries executed against relational databases use the current date and time of the database server.
Implicit timezone (implementation-defined)	Supported. The timezone is obtained from the Java Virtual Machine; however, queries executed against relational databases use the current timezone of the database server.
Available documents	Supported. Available documents include XML files accessed through http:, ftp:, and file: URI schemes using fn:doc(). See “XML Data Sources” on page 116 for rules governing URIs.
Available collections	Supported. Available collections are determined by which connections are configured. See “Specifying Relational Database Tables” on page 118 for details.
Default collection	Not supported.

Processing Model

[Table A-2](#) describes how the XQuery processing model relates to Stylus XQuery.

Table A-2. XQuery Processing Model

XQuery Language	Support
Data Model Generation	For a description of data model generation from XML, see “Data Model Representation of XML Documents” on page 117 . For a description of data model generation from relational sources, see “Data Model Representation of Relational Tables” on page 120 .
Schema import processing	Not supported.
Expression processing	Supported.
Serialization	Supported through XQJ. See Appendix D “Serialization Support” on page 439 for details.
Consistency constraints	Stylus XQuery adheres to the consistency constraints as listed in this section of the XQuery specification.

Error Handling

[Table A-3](#) describes how Stylus XQuery handles errors.

Table A-3. Error Handling

XQuery Language	Support
Error reporting	Supported. The static typing feature is also supported.

Concepts

Table A-4 describes how Stylus XQuery works with some key concepts defined by XQuery.

Table A-4. XQuery Documents

XQuery Language	Support
Document order	Supported.
Atomization	Supported.
Effective Boolean value	Supported.
Input sources	See fn:doc (15.5.4) and fn:collection (15.5.6) .
URI literals	Supported.

Types

Table A-5 describes how Stylus XQuery handles XQuery types.

Table A-5. XQuery Types

XQuery Language	Support
Predefined schema types	Supported. See “Supported XQuery Atomic Types” on page 459 for a list of supported atomic types.
Typed value and string value	Supported.
SequenceType syntax	Supported.
SequenceType matching	Supported. For queries evaluated against a relational database, cardinality testing is not performed.

Comments

Table A-6 describes how Stylus XQuery supports comments.

Table A-6. Comments

XQuery Language	Support
Comments	Supported.

3 Expressions

This section describes how Stylus XQuery supports the following XQuery expressions.

- “Primary Expressions” on page 348
- “Path Expressions” on page 349
- “Sequence Expressions” on page 350
- “Arithmetic Expressions” on page 350
- “Comparison Expressions” on page 351
- “Logical Expressions” on page 351
- “Constructors” on page 352
- “FLWOR Expressions” on page 353
- “Ordered and Unordered Expressions” on page 353
- “Conditional Expressions” on page 354
- “Quantified Expressions” on page 354
- “Expressions on SequenceTypes” on page 355
- “Validate Expressions” on page 356
- “Extension Expressions” on page 356

Primary Expressions

Table A-7 describes how Stylus XQuery supports XQuery primary expressions.

Table A-7. XQuery Primary Expressions

XQuery Language	Support
Literals)	<p>Supported.</p> <p>For relational sources, the default encoding for string literals is the encoding used by the database. You can set it to Unicode using the option declaration <code>sql-unicode-literals</code>.</p> <p>By default, the precision and scale for decimals is:</p> <p>DB2: 30, 15 Informix: 32, 15 Oracle: No default PostgreSQL: No default Microsoft SQL Server: 38,19 Sybase: 38, 19</p> <p>You can override the database default by using the <code>sql-decimal-cast</code> option declaration. See “Option Declarations” on page 275 for details. Also, the following constructor function creates a decimal that allows you to set the precision and scale:</p> <pre>ddtek:decimal (\$arg as xs:anyAtomicType?, \$precision as xs:integer, \$scale as xs:integer) as xs:decimal?</pre>
Variable references	Supported.
Parenthesized expressions	Supported.
Context item expression	Supported, except for initial context item.
Function calls	Supported.

Path Expressions

Table A-8 describes how Stylus XQuery supports path expressions.

Table A-8. XQuery Path Expressions

XQuery Language	Support
Steps	
Axis	Supported, except for the Full Axis feature.
Nodes tests	<p>Supported.</p> <p>For relational sources, name tests and kind tests that specify a name are not supported when the test is applied to a node with a computed name. In such cases, an error is raised. For example, the following XQuery raises an error:</p> <pre>for \$h in collection('holdings')/holdings/* return document { element {name(\$h)} {string(\$h)} } / stockholder</pre>
Predicates	Supported. Numeric predicates are compensated for relational sources. See “Understanding Compensation” on page 191 .
Unabbreviated syntax	Supported.
Abbreviated syntax	Supported.

Sequence Expressions

[Table A-9](#) describes how Stylus XQuery supports sequence expressions.

Table A-9. XQuery Sequence Expressions

XQuery Language	Support
Constructing sequences	Supported. Range expressions are compensated for relational sources. See “Understanding Compensation” on page 191 .
Filter expressions	Supported. Numeric predicates in filter expressions are compensated for relational sources. “Understanding Compensation” on page 191 .
Combining node sequences	Supported.

Arithmetic Expressions

[Table A-10](#) describes how Stylus XQuery supports arithmetic expressions.

Table A-10. XQuery Arithmetic Expressions

XQuery Language	Support
Arithmetic expressions	Supported. For relational sources, see Appendix B “Functions and Operators” on page 361 for restrictions.

Comparison Expressions

Table A-11 describes how Stylus XQuery supports comparison expressions.

Table A-11. XQuery Comparison Expressions	
XQuery Language	Support
Value comparisons	Supported.
General comparisons	Supported.
Node comparisons	Supported.

Logical Expressions

Table A-12 describes how Stylus XQuery supports logical expressions.

Table A-12. XQuery Logical Expressions	
XQuery Language	Support
Logical expressions	Supported.

Constructors

Table A-13 describes how Stylus XQuery supports constructors.

Table A-13. XQuery Constructors	
XQuery Language	Support
Direct element constructors	Supported.
Attributes	Supported.
Namespace declaration attributes	Supported.
Content	Supported.
Boundary whitespace	Supported.
Other direct constructors	Supported.
Computed element constructors	Supported.
Computed attribute constructors	Supported.
Document node constructors	Supported.
Text node constructors	Supported.
Computed processing instruction constructors	Supported.
Computed comment constructors	Supported.
In-scope namespaces of a constructed element	Supported. Stylus XQuery only supports inherit and preserve for the copy-namespaces mode.

FLWOR Expressions

Table A-14 describes how Stylus XQuery supports FLWOR expressions.

Table A-14. XQuery FLWOR Expressions

XQuery Language	Support
FLWOR expressions	Supported. For relational sources, positional variables are compensated. Also, relational sources do not support the collation in order by clauses. Any specified collation is ignored and sorting is performed according to the collation used by the database. See “ Specifying Collations ” on page 312 for more information about using the collation parameter. See “ Restructuring Data: FLWOR Expressions ” on page 84 for more information on using FLWOR expressions in your XQuery.

Ordered and Unordered Expressions

Table A-15 describes how Stylus XQuery supports ordered and unordered expressions.

Table A-15. XQuery Ordered and Unordered Expressions

XQuery Language	Support
Ordered expressions	Supported.
Unordered expressions	Supported.

Conditional Expressions

Table A-16 describes how Stylus XQuery supports conditional expressions.

Table A-16. XQuery Conditional Expressions	
XQuery Language	Support
Conditional expressions	Supported.

Quantified Expressions

Table A-17 describes how Stylus XQuery supports quantified expressions.

Table A-17. XQuery Quantified Expressions	
XQuery Language	Support
Quantified expressions	Supported.

Expressions on SequenceTypes

Table A-18 describes how Stylus XQuery supports expressions on SequenceTypes.

Table A-18. XQuery Expressions on Sequence Types	
XQuery Language	Support
Instance of	Supported. Compensated for relational sources. See “Understanding Compensation” on page 191 .
Typeswitch	Supported. Compensated for relational sources. See “Understanding Compensation” on page 191 .
Cast	Supported. See “Supported XQuery Atomic Types” on page 459 . For relational sources, not all data types are supported. See “Data Type Mappings” on page 447 for tables of supported database data types and information about how they are mapped to the XML schema data types. If the XML schema data type specified in a cast expression is not supported, the cast expression is compensated. See “Understanding Compensation” on page 191 .
Castable	Supported. Compensated for relational sources. See “Understanding Compensation” on page 191 .
Constructor functions	Supported. For relational sources, not all constructor functions are supported. See “5 Constructor Functions” on page 364 for which types are supported. If an XQuery expression specifies a constructor function that is not supported, the constructor function is compensated. See “Understanding Compensation” on page 191 .
Treat	Supported. Compensated for relational sources. See “Understanding Compensation” on page 191 .

Validate Expressions

Table A-19 describes how Stylus XQuery supports validate expressions.

Table A-19. XQuery Validate Expressions	
XQuery Language	Support
Validate expressions	Not supported. NOTE: Stylus XQuery supports a built-in validate function, <code>ddtek:validate</code> . See “ddtek:validate” on page 424 .

Extension Expressions

Table A-20 describes how Stylus XQuery supports extension expressions.

Table A-20. XQuery Extension Expressions	
XQuery Language	Support
Extension expressions	Supported. See “Using Extension Expressions” on page 285 for details.

4 Modules and Prologs

[Table A-21](#) describes how Stylus XQuery supports XQuery modules and prologs.

Table A-21. XQuery Modules and Prologs

XQuery Language	Support
Version declaration 1.1	Supported. The version can be 1.0 or 1.1. If an encoding declaration is specified, it is ignored. See “Specifying the XQuery Version” on page 60 for more information.
Module declaration	Supported.
Boundary-space declaration	Supported.
Default collation declaration	Supported. Any collation supported by the Java Virtual Machine can be specified. See “Specifying Collations” on page 312 for more information about collations.
Base URI declaration	Supported.
Construction declaration	Supported. Stylus XQuery always implements preserve.
Ordering mode declaration	Supported.
Empty order declaration	Supported.
Copy-namespaces declaration	Supported. Stylus XQuery always implements inherit, preserve.
Schema import	Not supported.
Decimal-format declaration 1.1	Supported for XQuery 1.1 only.
Module import	Supported. See “Library Module URI Resolvers” on page 296 .
Namespace declaration	Supported.
Default namespace declaration	Supported.
Variable declaration	Supported. 1.1 VarValue and VarDefaultValue declaration parts are supported in XQuery 1.1 only.

Table A-21. XQuery Modules and Prologs *(cont.)*

XQuery Language	Support
Function declaration	Supported.
Option declaration	Supported. See “Option Declarations” on page 275 for details.
Context item declaration	Supported for XQuery 1.1 only.
1.1	

5 Conformance

Stylus XQuery fulfills the requirements of XQuery Minimal Conformance.

Optional Features

[Table A-22](#) describes how Stylus XQuery supports optional XQuery features. Some optional features are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table A-22. XQuery Optional Features

XQuery Language	Support
Schema import feature	Not supported.
Schema validation feature	Not supported.
Static typing feature	Supported. See “Resolving Static Type Errors” on page 573 for more information about static typing.
Full axis feature	Supported. Compensated for relational sources.
Module feature	Supported. See “Library Module URI Resolvers” on page 296 .

Table A-22. XQuery Optional Features *(cont.)*

XQuery Language	Support
Serialization feature	Supported. See Appendix D “Serialization Support” on page 439 .
Trivial XML embedding feature	Not supported.

Data Model Conformance

Stylus XQuery supports the data types as described in [“Supported XQuery Atomic Types” on page 459](#). XML documents are mapped to the Data Model using Infoset mapping. Relational tables are mapped to the Data Model using SQL/XML mappings and PSVI mapping.

Namespaces

This section lists predefined namespaces.

Predefined Namespaces (Not Reserved)

[Table A-23](#) lists namespaces that are predefined, but not reserved.

Table A-23. Predefined Namespaces

Namespace Prefix	Namespace URI
xml	http://www.w3.org/XML/1998/namespace
xs	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
fn	http://www.w3.org/2005/xpath-functions
xdt	http://www.w3.org/2005/xpath-datatypes
local	http://www.w3.org/2005/xquery-local-functions

B Functions and Operators

This appendix describes how Stylus XQuery supports the following categories of XQuery functions and operators as described in the XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation 23 January 2007 located at: <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/>

In this appendix, numbered headings and items correspond to the sections in the [W3C Recommendation 23 January 2007](#) where that topic is discussed.

- “2 Accessors” on page 362
- “3 Error Function” on page 363
- “4 Trace Function” on page 363
- “5 Constructor Functions” on page 364
- “6 Functions and Operators on Numerics” on page 367
- “7 Functions on Strings” on page 369
- “8 Functions and Operators for anyURI” on page 372
- “9 Functions and Operators on Boolean Values” on page 373
- “10 Functions and Operators on Durations, Dates, and Times” on page 374
- “11 Functions Related to QNames” on page 381
- “12 Operators on base64Binary and hexBinary” on page 382
- “13 Functions and Operators on NOTATION” on page 383
- “14 Functions and Operators on Nodes” on page 383

- [“15 Functions and Operators on Sequences” on page 384](#)
- [“16 Context Functions” on page 387](#)
- [“17 Casting” on page 388](#)

2 Accessors

[Table B-1](#) describes how Stylus XQuery supports XQuery accessor functions. Some accessor functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-1. XQuery Accessor Functions

XQuery Function	Support
fn:node-name (2.1)	Supported.
fn:nilled (2.2)	Supported. Compensated for relational sources.
fn:string (2.3)	Supported.
fn:data (2.4)	Supported.
fn:base-uri (2.5)	Supported. Compensated for relational sources.
fn:document-uri (2.6)	Supported. Compensated for relational sources.

3 Error Function

[Table B-2](#) describes how Stylus XQuery supports the XQuery error function. The error function is compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-2. XQuery Error Function

XQuery Function	Support
fn:error (3)	Supported. Compensated for relational sources.

4 Trace Function

[Table B-3](#) describes how Stylus XQuery supports the XQuery trace function. The trace function is compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-3. XQuery Trace Function

XQuery Function	Support
fn:trace (4)	Supported. Compensated for relational sources.

5 Constructor Functions

Each predefined XQuery atomic type has an associated constructor function. Only those constructor functions that operate on supported atomic types are supported. See [“Supported XQuery Atomic Types” on page 459](#) for a list of the atomic types supported by Stylus XQuery.

5.1 Constructor Functions for XML Schema Built-in Types

[Table B-4](#) describes how Stylus XQuery supports the XQuery constructor functions for XML schema built-in types. Some of these constructor functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-4. XQuery Constructor Functions for XML Schema Built-In Types

XQuery Function	Support
xs:string (5.1)	Supported.
xs:boolean (5.1)	Supported.
xs:decimal (5.1)	Supported.
xs:float (5.1)	Supported. For relational sources, INF, -INF, and NaN are not supported.
xs:double (5.1)	Supported. For relational sources, INF, -INF, and NaN are not supported.
xs:duration (5.1)	Supported. Compensated for relational sources.
xs:dateTime (5.1)	Supported.
xs:time (5.1)	Supported.
xs:date (5.1)	Supported.
xs:gYearMonth (5.1)	Supported. Compensated for relational sources.
xs:gYear (5.1)	Supported. Compensated for relational sources.
xs:gMonthDay (5.1)	Supported. Compensated for relational sources.

Table B-4. XQuery Constructor Functions for XML Schema Built-In Types (cont.)

XQuery Function	Support
xs:gDay (5.1)	Supported. Compensated for relational sources.
xs:gMonth (5.1)	Supported. Compensated for relational sources.
xs:hexBinary (5.1)	Supported. Compensated for relational sources.
xs:base64Binary (5.1)	Supported. Compensated for relational sources.
xs:anyURI (5.1)	Supported. Compensated for relational sources.
xs:QName (5.1)	Supported.
xs:normalizedString (5.1)	Supported. Compensated for relational sources.
xs:token (5.1)	Supported. Compensated for relational sources.
xs:language (5.1)	Supported. Compensated for relational sources.
xs:NMTOKEN (5.1)	Supported. Compensated for relational sources.
xs:Name (5.1)	Supported. Compensated for relational sources.
xs:NCName (5.1)	Supported. Compensated for relational sources.
xs:ID (5.1)	Supported. Compensated for relational sources.
xs:IDREF (5.1)	Supported. Compensated for relational sources.
xs:ENTITY (5.1)	Supported. Compensated for relational sources.
xs:integer (5.1)	Supported.
xs:nonPositiveInteger (5.1)	Supported. Compensated for relational sources.
xs:negativeInteger (5.1)	Supported. Compensated for relational sources.
xs:long (5.1)	Supported.
xs:int (5.1)	Supported.
xs:short (5.1)	Supported.
xs:byte (5.1)	Supported.
xs:nonNegativeInteger (5.1)	Supported. Compensated for relational sources.
xs:unsignedLong (5.1)	Supported. Compensated for relational sources.
xs:unsignedInt (5.1)	Supported. Compensated for relational sources.
xs:unsignedShort (5.1)	Supported. Compensated for relational sources.
xs:unsignedByte (5.1)	Supported. Compensated for relational sources.
xs:positiveInteger (5.1)	Supported. Compensated for relational sources.
xs:yearMonthDuration (5.1)	Supported. Compensated for relational sources.

Table B-4. XQuery Constructor Functions for XML Schema Built-In Types *(cont.)*

XQuery Function	Support
xs:dayTimeDuration (5.1)	Supported. Compensated for relational sources.
xs:untypedAtomic (5.1)	Supported.

5.2 A Special Constructor Function for *xs:dateTime*

[Table B-5](#) describes how Stylus XQuery supports the Special Constructor Function for *xs:dateTime*.

Table B-5. A Special Constructor Function for *xs:dateTime*

XQuery Function	Support
fn:dateTime(\$arg1 as xs:date, \$arg2 as xs:time) as xs:dateTime (5.2)	Supported.

5.3 Constructor Functions for User-Defined Types

Not supported.

6 Functions and Operators on Numerics

This section describes how Stylus XQuery supports functions and operators on numeric data types.

6.2 Operators on Numeric Values

[Table B-6](#) describes how Stylus XQuery supports operators on numeric values.

Table B-6. XQuery Operators on Numeric Values

XQuery Operator	Support
op:numeric-add (6.2.1)	Supported.
op:numeric-subtract (6.2.2)	Supported.
op:numeric-multiply (6.2.3)	Supported.
op:numeric-divide (6.2.4)	Supported. For relational sources, an xs:float or xs:double value divided by zero raises an error instead of returning INF or -INF.
op:numeric-integer-divide (6.2.5)	Supported.
op:numeric-mod (6.2.6)	Supported.
op:numeric-unary-plus (6.2.7)	Supported.
op:numeric-unary-minus (6.2.8)	Supported.

6.3 Comparison of Numeric Values

Table B-7 describes how Stylus XQuery supports comparison operators on numeric values.

Table B-7. XQuery Comparison Operators on Numeric Values

XQuery Operator	Support
op:numeric-equal (6.3.1)	Supported.
op:numeric-less-than (6.3.2)	Supported.
op:numeric-greater-than (6.3.3)	Supported.

6.4 Functions on Numeric Values

Table B-8 describes how Stylus XQuery supports functions on numeric values. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-8. XQuery Functions on Numeric Values

XQuery Function	Support
fn:abs (6.4.1)	Supported.
fn:ceiling (6.4.2)	Supported. Compensated for Informix.
fn:floor (6.4.3)	Supported.
fn:round (6.4.4)	Supported.
fn:round-half-to-even (6.4.5)	Supported. Compensated for relational sources.

7 Functions on Strings

This section describes how Stylus XQuery supports functions on string data types.

7.2 Functions to Assemble and Disassemble Strings

[Table B-9](#) describes how Stylus XQuery supports functions to assemble and disassemble strings. All of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-9. XQuery Functions to Assemble and Disassemble Strings	
XQuery Function	Support
fn:codepoints-to-string (7.2.1)	Supported. Compensated for relational sources.
fn:string-to-codepoints (7.2.2)	Supported. Compensated for relational sources.

7.3 Equality and Comparison of Strings

[Table B-10](#) describes how Stylus XQuery supports functions that provide equality and comparison operations on strings. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-10. XQuery Functions for Equality and Comparison of Strings	
XQuery Function	Support
fn:compare (7.3.2)	Supported. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:codepoint-equal (7.3.3)	Supported. Compensated for relational sources.

7.4 Functions on String Values

Table B-11 describes how Stylus XQuery supports functions on string values. Some of these functions are compensated for relational sources; see “Understanding Compensation” on page 191 for details.

Table B-11. XQuery Functions on String Values

XQuery Function	Support
fn:concat (7.4.1)	Supported.
fn:string-join (7.4.2)	Supported. Compensated for relational sources.
fn:substring (7.4.3)	Supported.
fn:string-length (7.4.4)	Supported.
fn:normalize-space (7.4.5)	Supported. Compensated for relational sources.
fn:normalize-unicode (7.4.6)	Supported. Compensated for relational sources.
fn:upper-case (7.4.7)	Supported.
fn:lower-case (7.4.8)	Supported.
fn:translate (7.4.9)	Supported. Compensated for relational sources, except in this case: fn:translate(\$arg as xs:string?, \$mapString as xs:string, \$transString as xs:string) as xs:string fn:translate() is not compensated if all of the following conditions are true: <ul style="list-style-type: none">■ \$mapString and \$transString are literals.■ No characters of \$mapString appear in \$transString, and vice versa.■ The underlying database is not Sybase.
fn:escape-uri (7.4.10)	Supported. Compensated for relational sources.
fn:iri-to-uri (7.4.11)	Supported. Compensated for relational sources.
fn:escape-html-uri (7.4.12)	Supported. Compensated for relational sources.

7.5 Functions Based on Substring Matching

Table B-12 describes how Stylus XQuery supports functions based on substring matching.

Table B-12. XQuery Functions Based on Substring Matching

XQuery Function	Support
fn:contains (7.5.1)	Supported. Compensated for Informix, except when the second argument is a sting literal. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:starts-with (7.5.2)	Supported. Compensated for Informix, except when the second argument is a sting literal. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:ends-with (7.5.3)	Supported. Compensated for Informix, except when the second argument is a sting literal. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:substring-before (7.5.4)	Supported. Compensated for Informix. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:substring-after (7.5.5)	Supported. Compensated for Informix. See “Specifying Collations” on page 312 for information about using the collation parameter.

7.6 String Functions That Use Pattern Matching

Table B-13 describes how Stylus XQuery supports functions that use pattern matching. All of these functions are compensated for relational sources; see “Understanding Compensation” on page 191 for details.

Table B-13. String Functions That Use Pattern Matching

XQuery Function	Support
fn:matches (7.6.2)	Supported. Compensated for relational sources.
fn:replace (7.6.3)	Supported. Compensated for relational sources, except if all of the following conditions are true: <ul style="list-style-type: none">▪ <code>\$pattern</code> and <code>\$replacement</code> are string literals.▪ <code>\$pattern</code> is not a regular expression, does not contain unescaped characters: <code>([] * ? + ^ \$</code>.▪ <code>\$replacement</code> cannot include unescaped <code>\</code> or <code>\$</code> characters.▪ The underlying database is not Sybase.
fn:tokenize (7.6.4)	Supported. Compensated for relational sources.

8 Functions and Operators for anyURI

Table B-14 describes how Stylus XQuery supports functions on the anyURI data type. The resolve-uri function is compensated for relational sources; see “Understanding Compensation” on page 191 for details.

Table B-14. XQuery anyURI Functions

XQuery Function	Support
fn:resolve-uri (8.1)	Supported. Compensated for relational sources.

9 Functions and Operators on Boolean Values

This section describes how Stylus XQuery supports functions and operators on the boolean data type.

9.1 Boolean Constructor Functions

[Table B-15](#) describes how Stylus XQuery supports boolean constructor functions.

Table B-15. XQuery Boolean Constructor Functions	
XQuery Function	Support
fn:true (9.1.1)	Supported.
fn:false (9.1.2)	Supported.

9.2 Operators on Boolean Values

[Table B-16](#) describes how Stylus XQuery supports operators on boolean values.

Table B-16. XQuery Operators on Boolean Values	
XQuery Operator	Support
op:boolean-equal (9.2.1)	Supported.
op:boolean-less-than (9.2.2)	Supported.
op:boolean-greater-than (9.2.3)	Supported.

9.3 Functions on Boolean Values

Table B-17 describes how Stylus XQuery supports functions on boolean values.

Table B-17. XQuery Functions on Boolean Values

XQuery Function	Support
fn:not (9.3.1)	Supported.

10 Functions and Operators on Durations, Dates, and Times

This section describes how Stylus XQuery supports functions and operators on duration, date, and time data types.

10.1 and 10.2 Duration, Date, and Time Types

Table B-18 describes how Stylus XQuery supports functions on duration, date, and time data types. Some of these are compensated for relational sources, see [“Understanding Compensation” on page 191](#) for details.

Table B-18. Functions on Duration, Date, and Time Data Types

XQuery Function	Support
xs:dateTime (10.1)	Supported. See comments for xs:dateTime (5.1) .
xs:date (10.1)	Supported.
xs:time (10.1)	Supported. See comments for xs:time (5.1) .
xs:gYearMonth (10.1)	Supported. Compensated for relational sources.
xs:gYear (10.1)	Supported. Compensated for relational sources.

Table B-18. Functions on Duration, Date, and Time Data Types

XQuery Function	Support
xs:gMonthDay (10.1)	Supported. Compensated for relational sources.
xs:gMonth (10.1)	Supported. Compensated for relational sources.
xs:gDay (10.1)	Supported. Compensated for relational sources.

10.3 Two Totally Ordered Subtypes of Duration

Stylus XQuery supports the xs:yearMonthDuration (10.3.1) and xs:dayTimeDuration (10.3.2). For relational sources, these two subtypes are compensated. See [“Understanding Compensation” on page 191](#).

10.4 Comparisons of Duration, Date, and Time Values

[Table B-19](#) describes how Stylus XQuery supports comparisons of duration, date, and time values. Some of the comparison operators on duration, date, and time values are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-19. XQuery Comparisons of Duration, Date, and Time Values

XQuery Operator	Support
op:yearMonthDuration-equal (10.4.1)	Supported. Compensated for relational sources.
op:yearMonthDuration-less-than (10.4.2)	Supported. Compensated for relational sources.
op:yearMonthDuration-greater-than (10.4.3)	Supported. Compensated for relational sources.

Table B-19. XQuery Comparisons of Duration, Date, and Time Values *(cont.)*

XQuery Operator	Support
op:dayTimeDuration-equal (10.4.4)	Supported. Compensated for relational sources.
op:dayTimeDuration-less-than (10.4.5)	Supported. Compensated for relational sources.
op:dayTimeDuration-greater-than (10.4.6)	Supported. Compensated for relational sources.
op:duration-equal (10.4.7)	Supported. Compensated for relational sources.
op:dateTime-equal (10.4.8)	Supported.
op:dateTime-less-than (10.4.9)	Supported.
op:dateTime-greater-than (10.4.10)	Supported.
op:date-equal (10.4.11)	Supported.
op:date-less-than (10.4.12)	Supported.
op:date-greater-than (10.4.13)	Supported.
op:time-equal (10.4.14)	Supported.
op:time-less-than (10.4.15)	Supported.
op:time-greater-than (10.4.16)	Supported.
op:gYearMonth-equal (10.4.17)	Supported. Compensated for relational sources.
op:gYear-equal (10.4.18)	Supported. Compensated for relational sources.
op:gMonthDay-equal (10.4.19)	Supported. Compensated for relational sources.
op:gMonth-equal (10.4.20)	Supported. Compensated for relational sources.
op:gDay-equal (10.4.21)	Supported. Compensated for relational sources.

10.5 Component Extraction Functions on Durations, Dates, and Times

Table B-20 describes how Stylus XQuery supports component extraction functions. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-20. XQuery Component Extraction Functions

XQuery Function	Support
fn:years-from-duration (10.5.1)	Supported. Compensated for relational sources.
fn:months-from-duration (10.5.2)	Supported. Compensated for relational sources.
fn:days-from-duration (10.5.3)	Supported. Compensated for relational sources.
fn:hours-from-duration (10.5.4)	Supported. Compensated for relational sources.
fn:minutes-from-duration (10.5.5)	Supported. Compensated for relational sources.
fn:seconds-from-duration (10.5.6)	Supported. Compensated for relational sources.
fn:year-from-dateTime (10.5.7)	Supported.
fn:month-from-dateTime (10.5.8)	Supported.
fn:day-from-dateTime (10.5.9)	Supported.
fn:hours-from-dateTime (10.5.10)	Supported. Compensated for Informix.
fn:minutes-from-dateTime (10.5.11)	Supported. Compensated for Informix.
fn:seconds-from-dateTime (10.5.12)	Supported. Compensated for Informix.
fn:timezone-from-dateTime (10.5.13)	Supported. Compensated for relational sources.
fn:year-from-date (10.5.14)	Supported.
fn:month-from-date (10.5.15)	Supported.
fn:day-from-date (10.5.16)	Supported.
fn:timezone-from-date (10.5.17)	Supported. Compensated for relational sources.
fn:hours-from-time (10.5.18)	Supported. Compensated for Informix.
fn:minutes-from-time (10.5.19)	Supported. Compensated for Informix.
fn:seconds-from-time (10.5.20)	Supported. Compensated for Informix.
fn:timezone-from-time (10.5.21)	Supported. Compensated for relational sources.

10.6 Arithmetic Operators on Durations

Table B-21 describes how Stylus XQuery supports arithmetic operators on durations. All of these operators are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-21. XQuery Arithmetic Operators on Durations

XQuery Operator	Support
op:add-yearMonthDurations (10.6.1)	Supported. Compensated for relational sources.
op:subtract-yearMonthDurations (10.6.2)	Supported. Compensated for relational sources.
op:multiply-yearMonthDuration (10.6.3)	Supported. Compensated for relational sources.
op:divide-yearMonthDuration (10.6.4)	Supported. Compensated for relational sources.
op:divide-yearMonthDuration-by-yearMonthDuration (10.6.5)	Supported. Compensated for relational sources.
op:add-dayTimeDurations (10.6.6)	Supported. Compensated for relational sources.
op:subtract-dayTimeDurations (10.6.7)	Supported. Compensated for relational sources.
op:multiply-dayTimeDuration (10.6.8)	Supported. Compensated for relational sources.
op:divide-dayTimeDuration (10.6.9)	Supported. Compensated for relational sources.
op:divide-dayTimeDuration-by-dayTimeDuration (10.6.10)	Supported. Compensated for relational sources.

10.7 Timezone Adjustment on Dates and Time Values

Table B-22 describes how Stylus XQuery supports functions and operators used to adjust timezones on dateTime, date, and time values. All of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-22. Functions for Timezone Adjustment on dateTime, date, and time Values

XQuery Function	Support
fn:adjust-dateTime-to-timezone (10.7.1)	Supported. Compensated for relational sources.
fn:adjust-date-to-timezone (10.7.2)	Supported. Compensated for relational sources.
fn:adjust-time-to-timezone (10.7.3)	Supported. Compensated for relational sources.

10.8 Arithmetic Operators on Durations, Dates, and Times

Table B-23 describes how Stylus XQuery supports operators used to add and subtract durations from dateTime, date, and time. All of these operators are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-23. Operators for Adding and Subtracting Durations from dateTime, date, and time

XQuery Operator	Support
op:subtract-dateTimes (10.8.1)	Supported. Compensated for relational sources.
op:subtract-dates (10.8.2)	Supported. Compensated for relational sources.

Table B-23. Operators for Adding and Subtracting Durations from *dateTime*, *date*, and *time* (cont.)

XQuery Operator	Support
op:subtract-times (10.8.3)	Supported. Compensated for relational sources.
op:add-yearMonthDuration-to-dateTime (10.8.4)	Supported. Compensated for relational sources.
op:add-dayTimeDuration-to-dateTime (10.8.5)	Supported. Compensated for relational sources.
op:subtract-yearMonthDuration-from-dateTime (10.8.6)	Supported. Compensated for relational sources.
op:subtract-dayTimeDuration-from-dateTime (10.8.7)	Supported. Compensated for relational sources.
op:add-yearMonthDuration-to-date (10.8.8)	Supported. Compensated for relational sources.
op:add-dayTimeDuration-to-date (10.8.9)	Supported. Compensated for relational sources.
op:subtract-yearMonthDuration-from-date (10.8.10)	Supported. Compensated for relational sources.
op:subtract-dayTimeDuration-from-date (10.8.11)	Supported. Compensated for relational sources.
op:add-dayTimeDuration-from-date (10.8.12)	Supported. Compensated for relational sources.
op:subtract-dayTimeDuration-from-time (10.8.13)	Supported. Compensated for relational sources.

11 Functions Related to QNames

This section describes how Stylus XQuery supports functions related to QNames.

11.1 Constructor Functions for QNames

[Table B-24](#) describes how Stylus XQuery supports constructor functions for QNames. Some of these constructor functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-24. Constructor Functions for QNames

XQuery Function	Support
fn:resolve-QName (11.1.1)	Supported. Compensated for relational sources.
fn:QName (11.1.2)	Supported.

11.2 Operators and Functions Related to QNames

[Table B-25](#) describes how Stylus XQuery supports the operators and functions related to QNames. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-25. XQuery Operators and Functions Related to QNames

XQuery Operator or Function	Support
op:QName-equal (11.2.1)	Supported.
fn:prefix-from-Qname (11.2.2)	Supported.
fn:local-name-from-QName (11.2.3)	Supported.

Table B-25. XQuery Operators and Functions Related to QName *(cont.)*

XQuery Operator or Function	Support
fn:namespace-uri-from-QName (11.2.4)	Supported.
fn:namespace-uri-for-prefix (11.2.5)	Supported. Compensated for relational sources.
fn:in-scope-prefixes (11.2.6)	Supported. Compensated for relational sources.

12 Operators on base64Binary and hexBinary

[Table B-26](#) describes how Stylus XQuery supports comparisons of base64Binary and hexBinary values. One of these operators is compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-26. XQuery Comparisons of base64Binary and hexBinary Values

XQuery Operator	Support
op:hexBinary-equal (12.1.1)	Supported.
op:base64Binary-equal (12.1.2)	Supported. Compensated for relational sources.

I3 Functions and Operators on NOTATION

Table B-27 describes how Stylus XQuery supports operators that work with NOTATION.

Table B-27. XQuery Operators on NOTATION

XQuery Operator	Support
op:NOTATION-equal (13.1.1)	Supported.

I4 Functions and Operators on Nodes

Table B-28 describes how Stylus XQuery supports functions and operators on nodes. One of these functions is compensated for relational sources; see “Understanding Compensation” on page 191 for details.

Table B-28. XQuery Functions and Operators on Nodes

XQuery Function or Operator	Support
fn:name (14.1)	Supported.
fn:local-name (14.2)	Supported.
fn:namespace-uri (14.3)	Supported.
fn:number (14.4)	Supported.
fn:lang (14.5)	Supported. Compensated for relational sources.
op:is-same-node (14.6)	Supported.
op:node-before (14.7)	Supported.
op:node-after (14.8)	Supported.
fn:root (14.9)	Supported.

15 Functions and Operators on Sequences

This section describes how Stylus XQuery supports functions and operators on sequences.

15.1 General Functions and Operators on Sequences

[Table B-29](#) describes how Stylus XQuery supports functions and operators on sequences. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-29. XQuery General Functions and Operators on Sequences

XQuery Function or Operator	Support
fn:boolean (15.1.1)	Supported.
op:concatenate (15.1.2)	Supported
fn:index-of (15.1.3)	Supported. See “Specifying Collations” on page 312 for information about using the collation parameter. Compensated for relational sources.
fn:empty (15.1.4)	Supported.
fn:exists (15.1.5)	Supported.
fn:distinct-values (15.1.6)	Supported. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:insert-before (15.1.7)	Supported. Compensated for relational sources.
fn:remove (15.1.8)	Supported. Compensated for relational sources.
fn:reverse (15.1.9)	Supported. Compensated for relational sources.
fn:subsequence (15.1.10)	Supported. Compensated for relational sources.
fn:unordered (15.1.11)	Supported. Compensated for relational sources.

15.2 Functions that Test Cardinality of Sequences

[Table B-30](#) describes how Stylus XQuery supports functions that test cardinality of sequences. All of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-30. XQuery Functions that Test Cardinality on Sequences

XQuery Function	Support
fn:zero-or-one (15.2.1)	Supported. Compensated for relational sources.
fn:one-or-more (15.2.2)	Supported. Compensated for relational sources.
fn:exactly-one (15.2.3)	Supported. Compensated for relational sources.

15.3 Equals, Union, Intersection, and Except

[Table B-31](#) describes how Stylus XQuery supports functions and operators on equals, union, intersection, and except. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-31. XQuery Functions and Operators on Equals, Union, Intersection, and Except

XQuery Function or Operator	Support
fn:deep-equal (15.3.1)	Supported. See “Specifying Collations” on page 312 for information about using the collation parameter. Compensated for relational sources.
op:union (15.3.2)	Supported.
op:intersect (15.3.3)	Supported.
op:except (15.3.4)	Supported.

15.4 Aggregate Functions

[Table B-32](#) describes how Stylus XQuery supports aggregate functions.

Table B-32. XQuery Aggregate Functions

XQuery Function	Support
fn:count (15.4.1)	Supported.
fn:avg (15.4.2)	Supported.
fn:max (15.4.3)	Supported. See “Specifying Collations” on page 312 for information about using the collation parameter.
fn:min (15.4.4)	
fn:sum (15.4.5)	Supported.

15.5 Functions and Operators That Generate Sequences

[Table B-33](#) describes how Stylus XQuery supports functions and operators that generate sequences. Some of these functions and operators are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-33. XQuery Functions and Operators That Generate Sequences

XQuery Function or Operator	Support
op:to (15.5.1)	Supported. Compensated for relational sources.
fn:id (15.5.2)	Supported. Compensated for relational sources.
fn:idref (15.5.3)	Supported. Compensated for relational sources.
fn:doc (15.5.4)	Supported. See “XML Data Sources” on page 116 for rules governing URIs.

Table B-33. XQuery Functions and Operators That Generate Sequences (*cont.*)

XQuery Function or Operator	Support
fn:doc-available (15.5.5)	Supported.
fn:collection (15.5.6)	Supported.

16 Context Functions

Table B-34 describes how Stylus XQuery supports context functions. Some of these functions are compensated for relational sources; see [“Understanding Compensation” on page 191](#) for details.

Table B-34. XQuery Context Functions

XQuery Function	Support
fn:position (16.1)	Supported. Compensated for relational sources.
fn:last (16.2)	Supported. Compensated for relational sources.
fn:current-dateTime (16.3)	Supported. For relational sources, values are returned without timezones.
fn:current-date (16.4)	Supported. For relational sources, values are returned without timezones.
fn:current-time (16.5)	Supported. For relational sources, values are returned without timezones.
fn:implicit-timezone (16.6)	Supported. Compensated for relational sources.
fn:default-collation (16.7)	Supported. Compensated for relational sources.
fn:static-base-uri (16.8)	Supported. Compensated for relational sources.

I7 Casting

Supported.

C Built-in Functions and Options

This appendix describes Stylus XQuery built-in functions and options. Unless stated otherwise, all functions and options described in this chapter are supported for XQuery 1.1 and XQuery 1.0.

This appendix contains the following sections:

- [“Stylus XQuery Built-In Functions” on page 389](#)
- [“HTTP Functions <request> Element” on page 433](#)
- [“Stylus XQuery Options” on page 437](#)
- [“Namespaces” on page 437](#)

Stylus XQuery Built-In Functions

This section describes the following Stylus XQuery built-in functions:

- [ddtek:analyze-edi-from-string](#)
- [ddtek:analyze-edi-from-url](#)
- [ddtek:convert-to-xml](#)
- [ddtek:decimal](#)
- [ddtek:edi-to-xml-from-string](#)
- [ddtek:edi-to-xml-from-url](#)
- [ddtek:format-date](#)
- [ddtek:format-date-time](#)
- [ddtek:format-number](#)
- [ddtek:format-time](#)
- [ddtek:http-delete](#)
- [ddtek:http-get](#)
- [ddtek:http-head](#)

- [ddtek:http-options](#)
- [ddtek:http-post](#)
- [ddtek:http-put](#)
- [ddtek:http-trace](#)
- [ddtek:info](#)
- [ddtek:isValid](#)
- [ddtek:javaCast](#)
- [ddtek:ltrim](#), [ddtek:rtrim](#), and [ddtek:trim](#)
- [ddtek:parse](#)
- [ddtek:serialize](#)
- [ddtek:serialize-to-url](#)
- [ddtek:sql-delete](#)
- [ddtek:sql-insert](#)
- [ddtek:sql-update](#)
- [ddtek:validate](#)
- [ddtek:validate-and-report](#)
- [ddtek:wscall](#)

In addition, Stylus XQuery supports the DB2 V9.1 for Linux/UNIX/Windows, Microsoft SQL Server 2005, and Oracle 10g R2 XQuery extensions through a set of built-in (predeclared) XQuery functions that map one-to-one to the database features. See [“Querying XML on Microsoft SQL Server 2005” on page 497](#) and [“Querying XML on Oracle” on page 490](#) for examples.

ddtek:analyze-edi-from-string

[ddtek:analyze-edi-from-string](#) analyzes an EDI stream and provides an error report that describes any errors that will prevent EDI messages from being converted to XML using the [ddtek:edi-to-xml-from-string](#) function. The report generated by [ddtek:analyze-edi-from-string](#) is used by [ddtek:edi-to-xml-from-string](#) to filter messages that contain errors, allowing processing of EDI sources that contain errors.

This function is supported only for EDIFACT, HIPAA, and X12 EDI dialects.

Function Declaration

```
declare function ddtek:analyze-edi-from-string(  
  $url as xs:string,  
  $edi as xs:string)  
  as document-node(element(*, xs:untyped)) external;
```

where:

\$url specifies the properties you want the EDI conversion engine to use when converting the EDI stream to XML – EDI:tbl=yes, for example.

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

\$edi can be any EDI document in a supported dialect specified as a string.

Note: ddtek:analyze-edi-from-string requires that XML Converters are reachable in the current classpath and that a valid XML Converters license is available.

See [“Analyzing EDI to XML Conversions” on page 299](#) for more information.

ddtek:analyze-edi-from-url

ddtek:analyze-edi-from-url analyzes an EDI stream and provides an error report that describes any errors that will prevent EDI

messages from being converted to XML using the [ddtek:edi-to-xml-from-url](#) function. The report generated by `ddtek:analyze-edi-from-url` is used by [ddtek:edi-to-xml-from-url](#) to filter messages that contain errors, allowing processing of EDI sources that contain errors.

This function is supported only for EDIFACT, HIPAA, and X12 EDI dialects.

Function Declaration

```
declare function ddtek:analyze-edi-from-url(
  $url as xs:string)
  as document-node(element(*, xs:untyped)) external;
```

where:

\$url specifies the full path of the EDI source you want to convert, as well as the properties you want the EDI conversion engine to use when converting the EDI stream to XML – `EDI:tbl=yes?file:///c:/EDI/code99.x12`, for example.

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

Note: `ddtek:analyze-edi-from-url` requires that XML Converters are reachable in the current classpath and that a valid XML Converters license is available.

See [“Analyzing EDI to XML Conversions” on page 299](#) for more information.

ddtek:convert-to-xml

ddtek:convert-to-xml allows you to convert non-XML data passed as xs:string into XML using Stylus XML Converters expressed using the converter URI syntax. The source for the XML data might have been extracted from a database, XML document, or other sources.

Note: ddtek:convert-to-xml requires that XML Converters are reachable in the current classpath and that a valid XML Converters license is available.

Function Declaration

```
ddtek:convert-to-xml($input as xs:string, $options as xs:string) as  
document-node(element(*,xs:untyped))
```

where:

\$input is a character string

\$options are properties used by the conversion engine when converting to XML. Separate options with a colon (long=yes:tbl=no, for example).

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

Example Using the EDI Converter

This example shows an EDI message being converted to XML using the EDI converter:

```
let $edimessage := "ISA+00+ +00+ +01+1515151515"
```

```

+01+5151515151 +041201+1217+^+00403+000032123+0+P+*'
GS+CT+9988776655+1122334455+20041201+1217+128+X+004030'
ST+831+00128001'
BGN+00+88200001+20041201'
N9+BT+88200001'
TRN+1+88200001'
AMT+2+100000'
QTY+46+1'
SE+7+00128001'
GE+1+128'
IEA+1+000032123'"
return
  ddtেক:convert-to-xml($edimessage, "EDI:long=yes")

```

Example Using the CSV Converter

This example shows the CSV converter being used to convert a simple comma-separated values file (submitted as a string) to XML:

```

declare option ddtেক:serialize "indent=yes";
ddtek:convert-to-xml("a,b,c", "CSV")

```

ddtek:decimal

ddtek:decimal allows you to specify precision and scale of a decimal value. For example:

```

ddtek:decimal($v as xs:anyAtomicType?, $p as xs:integer, $s as xs:integer)
  as xs:decimal?

```

ddtek:edi-to-xml-from-string

ddtek:edi-to-xml-from-string allows you to convert an EDI stream to XML when the EDI is stored in memory as a String datatype. The report used as an input to this function, which is generated by [ddtek:analyze-edi-from-string](#), filters out errors in the EDI

source that would otherwise cause the conversion to fail. This function is supported only for EDIFACT, HIPAA, and X12 EDI dialects.

Function Declaration

```
declare function ddtek:edi-to-xml-from-string(  
  $url as xs:string,  
  $edi as xs:string,  
  $report as document-node(element(*, xs:untyped)))  
  as document-node(element(*, xs:untyped)) external;
```

where:

\$url specifies the properties you want the EDI conversion engine to use when converting the EDI stream to XML – EDI:tbl=yes, for example.

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

\$edi is any EDI document in a supported dialect.

\$report is the report generated by the [ddtek:analyze-edi-from-string](#) function.

Note: ddtek:edi-to-xml-from-string requires that XML Converters are reachable in the current classpath and that a valid XML Converters license is available.

See [“Analyzing EDI to XML Conversions” on page 299](#) for more information.

ddtek:edi-to-xml-from-url

ddtek:analyze-edi-from-url allows you to convert an EDI stream to XML. The report used as an input to this function, which is generated by [ddtek:analyze-edi-from-url](#), filters out errors in the EDI source that would otherwise cause the conversion to fail. This function is supported only for EDIFACT, HIPAA, and X12 EDI dialects.

Function Declaration

```
declare function ddtek:edi-to-xml-from-url(
  $url as xs:string ,
  $report as document-node(element(*, xs:untyped)) )
  as document-node(element(*, xs:untyped)) external;
```

where:

\$url specifies the full path of the EDI source you want to convert, as well as the properties you want the EDI conversion engine to use when converting the EDI stream to XML – EDI:tbl=yes?file:///c:/EDI/code99.x12, for example.

To learn more about conversion properties, see the section "EDI XML Converter Properties" in the *Stylus XML Converters User's Guide and Reference* manual.

\$report is the report generated by the [ddtek:analyze-edi-from-url](#) function.

Note: ddtek:edi-to-xml-from-url requires that XML Converters are reachable in the current classpath and that a valid XML Converters license is available.

See “[Analyzing EDI to XML Conversions](#)” on page 299 for more information.

ddtek:format-date

The `ddtek:format-date` function can be used to format date strings. Depending on your requirements, the [ddtek:format-date-time](#) function can also be used for this purpose.

The `ddtek:format-date` function is based on XSLT 2.0 date formatting functions as defined in the XSL Transformations (XSLT) Version 2.0 W3C Recommendation 23 January 2007. See <http://www.w3.org/TR/xslt20/#format-date> for more information.

Function Declarations

```
declare function ddtek:format-date($value as xs:date?, $picture as xs:string)
as xs:string?
```

```
declare function ddtek:format-date(
  $value as xs:date?,
  $picture as xs:string,
  $language as xs:string?,
  $calendar as xs:string?,
  $country as xs:string?) as xs:string?
```

where:

\$value is the date string to be formatted.

\$picture is a sequence of variable markers and literal substrings used to specify formatting of *\$value*. See “[Picture String](#)” for more information on specifying the *\$picture* argument.

\$language is the language to be used for the result of the `ddtek:format-date` function. It is used for names (days, for

example), numbers when expressed as words, hour convention, and the first day of the week and month of the year (Sunday versus Monday, for example). If the language is not specified, Stylus XQuery uses the language specified for the Java Virtual Machine (JVM) where Stylus XQuery is installed.

\$calendar is the type of calendar whose conventions are to be used to convert the string supplied in the *\$value* argument to a value in that calendar. If the calendar is not specified, Stylus XQuery uses the calendar specified for the Java Virtual Machine (JVM) where Stylus XQuery is installed.

\$country is the country in which the event represented by the date string occurs. Valid values are those specified by ISO 3166-1, ISO 3166-2, and ISO 3166-3. If the country is not specified, Stylus XQuery uses the country specified for the Java Virtual Machine (JVM) where Stylus XQuery is installed.

For more information on *\$language*, *\$calendar*, and *\$country* arguments, see <http://www.w3.org/TR/xslt20/#lang-cal-country> in the XSLT 2.0 specification.

Picture String

As described previously, the *\$picture* argument is a sequence of variable markers and literal substrings used to specify formatting of *\$value*. Variable markers are indicated using bracket pairs to surround a *specifier*; values not in brackets are taken as literal substrings. Consider the following picture string:

```
" [M] - [D] - [Y] "
```

Here, month (M), day (D), and year (Y) specifiers are separated by dash (-) literal substrings. Note that whitespace in literal substrings is preserved, so

```
" [M] - [D] - [Y] "
```

and

"[M] - [D] - [Y]"

yield different strings.

The following table summarizes commonly used specifiers:

Table 12-6. Common Picture String Specifiers

Specifier	Description
Y	Year
M	Month in year
D	Day in month
d	Day in year
f	Day of week
W	Week in year
w	Week in month
H	Hour in day (24 hours)
h	Hour in half-day (12 hours)
P	am/pm marker
m	Minute in hour
s	Second in minute
f	Fractional seconds
Z	Timezone as a time offset from Coordinated Universal Time (UTC); UTC+1, for example. Also accepts conventional timezone abbreviations (EST for Eastern Standard Timezone, for example)
z	Timezone as a time offset from Greenwich Mean Time (GMT)
C	Calendar (the name or abbreviation)
E	The name of a baseline for the numbering of years; the reign of a monarch, for example

For a complete description of the picture string, see <http://www.w3.org/TR/xslt20/#date-picture-string> in the XSLT 2.0 specification.

Examples

The following list shows pairs of `ddtek:format-date` function declarations and using an input value of `$d`, specified as `xs:date('2002-12-31')`. The complete XQuery code might look like this, for example:

```
let $d := xs:date('2002-12-31')
return (
  ddtek:format-date($d, "[Y0001]-[M01]-[D01]")
)
```

This XQuery returns this string:

2002-12-31

Here are other examples of the `ddtek:format-date` function and the XQuery result:

```
ddtek:format-date($d, "[M]-[D]-[Y]")
12-31-2002
```

```
ddtek:format-date($d, "[D]-[M]-[Y]")
31-12-2002
```

```
ddtek:format-date($d, "[D1] [MI] [Y]")
31 XII 2002
```

```
ddtek:format-date($d, "[D1o] [MNn], [Y]", "en", (), ())
31st December, 2002
```

```
ddtek:format-date($d, "[D01] [MN,*-3] [Y0001]", "en", (),
())
31 DEC 2002
```

```
ddtek:format-date($d, "[MNn] [D], [Y]", "en", (), ())
December 31, 2002
```

```
ddtek:format-date($d, "[D] [MNn], [Y]", "de", (), ())
```

31 Dezember, 2002

```
ddtek:format-date($d, "[FNn] [D] [MNn] [Y]", "sv", (), ())
tisdag 31 december 2002
```

```
ddtek:format-date($d, "[[Y0001]-[M01]-[D01]]")
[2002-12-31]
```

```
ddtek:format-date($d, "[YWw]", "en", (), ())
Two Thousand and Two
```

```
ddtek:format-date($d, "[Dwo] [MNn]", "de", (), ())
einunddreißigste Dezember
```

ddtek:format-date-time

The `ddtek:format-date-time` function can be used to format date-time strings. Depending on your requirements, the [ddtek:format-date](#) and [ddtek:format-time](#) functions can also be used for this purpose.

The `ddtek:format-date-time` function is based on XSLT 2.0 date formatting functions as defined in the XSL Transformations (XSLT) Version 2.0 W3C Recommendation 23 January 2007. See <http://www.w3.org/TR/xslt20/#format-date> for more information.

Function Declarations

```
declare function ddtek:format-dateTime($value as xs:dateTime?,
$picture as xs:string) as xs:string?
```

```
declare function ddtek:format-dateTime(
$value as xs:dateTime?,
$picture as xs:string,
$language as xs:string?,
$calendar as xs:string?,
$country as xs:string?) as xs:string?
```

See [ddtek:format-date](#) for a discussion of function arguments.

Examples

The following list shows pairs of `ddtek:format-dateTime` function declarations and using an input value of `$dt`, specified as

`xs:dateTime('2002-12-31T15:58:00')`. The complete XQuery code might look like this, for example:

```
let $dt := xs:dateTime('2002-12-31T15:58:00')
return (
  ddtek:format-dateTime($dt, "[h].[m01][Pn] on [FNn], [D1o]
[MNn]")
)
```

This XQuery returns this string:

3.58p.m. on Tuesday, 31st December

Here is another example of the `ddtek:format-dateTime` function and the XQuery result:

```
ddtek:format-dateTime($dt, "[M01]/[D01]/[Y0001] at [H01]:[m01]:[s01]")
12/31/2002 at 15:58:00
```

ddtek:format-number

The `ddtek:format-number` function is used to format numbers.

The `ddtek:format-number` function is based on XSLT 2.0 `format-number` function as defined in the XSL Transformations (XSLT) Version 2.0 W3C Recommendation 23 January 2007. See <http://www.w3.org/TR/xslt20/#format-number> for more information.

Function Declarations

```
declare function ddtek:format-number($number as numeric?, $format
as xs:string) as xs:string

declare function ddtek:format-number(
```

```

$number as numeric?,
$format as xs:string,
$decimal-separator as xs:string?,
$grouping-separator as xs:string?,
$infinity as xs:string?,
$minus-sign as xs:string?,
$NaN as xs:string?,
$percent as xs:string?,
$per-mille as xs:string?,
$zero-digit as xs:string?,
$digit as xs:string?,
$pattern-separator as xs:string?) as xs:string

```

where:

\$number is the string representing the number you want to format.

\$format is the format to be used for the digit representation. The default is #, but if you use the second function declaration you can use the value specified for the digit parameter.

The second function declaration takes additional parameters, as summarized in the following table.

Table 12-7. *ddtek:format-number* Function Parameters

Parameter	Value	Description
decimal-separator	char	Character used as the decimal character. Default is ".".
grouping-separator	char	Character used as the thousands separator. Default is ",".
infinity	string	String used to represent infinity. Default is "Infinity".
minus-sign	char	Character used to indicate negative numbers. Default is "-".

Table 12-7. *ddtek:format-number* Function Parameters

Parameter	Value	Description
NaN	string	String used when the value is not a number. Default is "NaN".
percent	char	Character used as the percent sign. Default is "%".
per-mille	char	Character used as the per-thousand sign character. Default is "‰".
zero-digit	char	Character used as the zero digit. Default is "0".
digit	char	Character used to indicate a place where a digit is required. Default is "#".
pattern-separator	char	Character used to separate positive and negative subpatterns in a format pattern. Default is ";".

Note that when you use the second function declaration for `ddtek:format-number`, you must specify all parameters, even those for which you wish to use the default values. To indicate that you wish to use a default value, specify the parameter as `()`.

Examples

Here are a few paired examples that show how to use the `ddtek:format-number` function. The function is shown on the first line; the result is shown on the second line.

These examples use the first function declaration:

```
ddtek:format-number(500100, "#")
500100
```

```
ddtek:format-number(500100, "###,###.00")
500,100.00
```

```
ddtek:format-number(0.23456, "%")
23%
```

The following examples use the second function declaration. Here is an example that uses European style formatting for numbers – that is, using the comma for the decimal point and the period for the thousands separator:

```
ddtek:format-number(26825.8, "#.###,00", ",", ".", (), (), (), (), (), (), (), ()
(), ())
26.825,80
```

In this example, the *\$format* argument is defined using a custom digit format, D:

```
ddtek:format-number(123456789, "$DDD,DDD,DDD.DD", (), (), (), (), (), (), (), ()
(), 'D', ())
$123,456,789
```

Note that in both of these examples all parameters for the `ddtek:format-number` declaration are specified.

ddtek:format-time

The `ddtek:format-time` function can be used to format time strings. Depending on your requirements, the [ddtek:format-date-time](#) function can also be used for this purpose.

The `ddtek:format-time` function is based on XSLT 2.0 date formatting functions as defined in the XSL Transformations (XSLT) Version 2.0 W3C Recommendation 23 January 2007. See <http://www.w3.org/TR/xslt20/#format-date> for more information.

Function Declarations

```
declare function ddtek:format-time($value as xs:time?, $picture as
xs:string) as xs:string?
```

```
declare function ddtek:format-time(
$value as xs:time?,
$picture as xs:string,
$language as xs:string?,
$calendar as xs:string?,
$country as xs:string?) as xs:string?
```

See [ddtek:format-date](#) for a discussion of function arguments.

Examples

The following list shows pairs of `ddtek:format-time` function declarations and using an input value of `$t`, specified as `xs:time('15:58:00')`. The complete XQuery code might look like this, for example:

```
let $t := xs:time('15:58:00')
return (
  ddtek:format-time($t, "[h]:[m01] [PN]", "en", (), ())
)
```

This XQuery returns this string:

```
3.58 P.M.
```

Here are other examples of the `ddtek:format-time` function and the XQuery result:

```
ddtek:format-time($t, "[h]:[m01]:[s01] [Pn]", "en", (), ())
3:58:00 p.m.
```

```
ddtek:format-time($t, "[h]:[m01]:[s01] [PN] [ZN,*-3]", "en", (), ())
3:58:00 P.M.
```

```
ddtek:format-time($t, "[h]:[m01]:[s01] o'clock [PN] [ZN,*-3]", "en", (), ())
3:58:00 o'clock P.M.
```



```
ddtek:format-time($t,"[H01]:[m01]")
15:58

ddtek:format-time($t,"[H01]:[m01]:[s01].[f001]")
15:58:00.000

ddtek:format-time($t,"[H01]:[m01]:[s01] [z]", "en", (), ())
15:58:00

ddtek:format-time($t,"[H01]:[m01] Uhr [z]", "de", (), ())
15:58 Uhr
```

ddtek:http-delete

The `ddtek:http-delete` function requests that the server delete the resource identified by the request URI. This function can be overridden by human intervention (or other means) on the server. Because of this, the client cannot be guaranteed that the operation has been carried out, even if the status code returned from the server indicates that the action has been completed successfully. However, the server should not indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

Successful responses are:

- 200 (OK) – the response includes a response body describing the status
- 202 (Accepted) – the action has not yet been enacted
- 204 (No Content) – the action has been enacted but the response does not include a response body

Note that responses to this function are not cacheable.

If the request passes through a cache and the URI identifies one or more currently cached entities, those entries should be treated as stale.

Function Declarations

```
declare function ddtek:http-delete($url as xs:string) as
document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-delete($url as xs:string,
$request as element()?) as
document-node(element(*,xs:untyped)) external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions `<request>` Element” on page 433](#) for more information.

ddtek:http-get

The `ddtek:http-get` function retrieves whatever information (in the form of an entity) is identified by the Request-URI. For example, if the Request-URI refers to a data-producing process, it is the produced data that is returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). This reduces unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

If a Range header field is included, the request is for only the part of the entity specified by the range header. This allows partially retrieved entities to be completed without transferring previously received data.

Function Declarations

```
declare function ddtek:http-get($url as xs:string) as
document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-get($url as xs:string, $request
as element()?) as document-node(element(*,xs:untyped))
external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions <request> Element” on page 433](#) for more information.

ddtek:http-head

The `ddtek:http-head` function is identical to the GET method except that the server must not return a message-body in the response. The meta-information contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request. This allows a client to obtain meta-information about a resource without actually transferring the resource itself.

The head function is often used for testing hyperlinks, accessibility and for determining if a document has been recently modified.

When your program is implementing caching, it is important to note that if the HEAD response indicates that the cached entity differs from the current entity, such as by a change in the Content-Length, Content-MD5, ETag or Last-Modified, the cache must treat the cached entry as stale.

Function Declarations

```
declare function ddtek:http-head($url as xs:string) as
document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-head($url as xs:string,
$request as element()?) as
document-node(element(*,xs:untyped)) external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions <request> Element” on page 433](#) for more information.

ddtek:http-options

The `ddtek:http-options` function represents a request for information about the communication options available on the request/response chain identified by the request URI.

This function allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating resource retrieval.

Function Declarations

```
declare function ddtek:http-options($url as xs:string) as
document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-options($url as xs:string,
$request as element()?) as
document-node(element(*,xs:untyped)) external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$request is a set of options specified as <request> element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions <request> Element” on page 433](#) for more information.

ddtek:http-post

The ddtek:http-post function is used to request that the origin server (that is, the server hosting the resource) accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. Essentially this means that the POST data will be stored by the server and usually will be processed by a server side application.

The ddtek:http-post function is designed to allow a uniform approach to the following types of Web service application activities:

- Annotating existing resources
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
- Providing a block of data, such as the result of submitting a form, to a data-handling process

- Extending a database through an append operation

It is generally expected that a POST request will have some side effect on the server, such as writing to a database, and the HTTP specification suggests that user agents represent user actions which result in a POST request in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested. This however, is not a requirement.

Function Declarations

```
declare function ddtek:http-post($url as xs:string,
  $payload as item()?) as
  document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-post($url as xs:string,
  $payload as item()?, $request as element()?) as
  document-node(element(*,xs:untyped)) external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$payload is an element that contains the service response.

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions `<request>` Element” on page 433](#) for more information.

ddtek:http-put

The `ddtek:http-put` function requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, the enclosed entity should be considered as a modified version of the one residing on the origin server. If the URI does not point to an existing resource, and that URI is

capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.

If the request passes through a cache and the URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this function are not cacheable.

The fundamental difference between POST and PUT requests is reflected in the different meaning of the request URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request – the user agent knows what URI is intended and the server must not attempt to apply the request to some other resource.

Unless otherwise specified for a particular entity-header, the entity-headers in the PUT request should be applied to the resource created or modified by the PUT.

Function Declarations

```
declare function ddtek:http-put($url as xs:string, $payload
as item()?, $request as element()?) as
document-node(element(*,xs:untyped)) external;

declare function ddtek:http-put($url as xs:string, $payload
as item()?) as document-node(element(*,xs:untyped))
external;
```

where:

\$url is the URI of the origin server resource
(<http://www.examples.xquery.com>, for example).

\$payload is an element that contains the service response.

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions `<request>` Element” on page 433](#) for more information.

ddtek:http-trace

The `ddtek:http-trace` function is primarily used for debugging and testing purposes, and simply requests that the server echo back the request it received. This can be useful for identifying any changes to the request that is made by proxies.

The `http-trace` function is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request should reflect the message received back to the client as the entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a `max-Forwards` value of zero (0) in the request (see section 14.31 of RFC2616).

Function Declarations

```
declare function ddtek:http-trace($url as xs:string) as
document-node(element(*,xs:untyped)) external;
```

```
declare function ddtek:http-trace($url as xs:string,
$request as element()?) as
document-node(element(*,xs:untyped)) external;
```

where:

\$url is the URI of the origin server resource
(`http://www.examples.xquery.com`, for example)

\$request is a set of options specified as `<request>` element attributes to be consumed by the origin server (a username and password, for example). See [“HTTP Functions `<request>` Element” on page 433](#) for more information.

ddtek:info

Simple function that returns Stylus XQuery version and build information and Java system properties for your Stylus XQuery installation.

The function declaration for ddtek:info is:

```
declare function ddtek:info() as document-node() external;
```

The ddtek:info function returns <ddxq> and <java> elements, a sample of which is shown here:

```
<info xmlns="">
  <ddxq>

    <name>Stylus XQuery</name>
    <version>4.0</version>
    <build>R0693</build>
    <build-date>Wed May 27 20:03:55 CEST 2009</build-date>
  </ddxq>
  <java>
    <java.runtime.name>Java(TM) SE Runtime Environment</java.runtime.name>
    <sun.boot.library.path>C:\Program
Files\Java\jre1.6.0_03\bin</sun.boot.library.path>
    <java.vm.version>1.6.0_03-b05</java.vm.version>
    <java.vm.vendor>Sun Microsystems Inc.</java.vm.vendor>
    <java.vendor.url>http://java.sun.com/</java.vendor.url>
    <path.separator>;</path.separator>
    <java.vm.name>Java HotSpot(TM) Client VM</java.vm.name>
    <file.encoding.pkg>sun.io</file.encoding.pkg>
    <user.country>US</user.country>
    <sun.os.patch.level>Service Pack 3</sun.os.patch.level>
    ...
  </java>
</info>
```

NOTE: The specific subelements of the <java> element vary based on your installation.

ddtek:isValid

This function is similar to [ddtek:validate](#) but instead of returning \$arg1 unchanged, it returns a boolean indicating whether it is valid for the specified XML Schema. Unlike ddtek:validate, ddtek:isValid does not cause the XQuery execution to throw an exception when the validation fails. An exception is thrown, however, if \$schema does not resolve to a valid XML Schema resource.

The function declaration for ddtek:isValid is:

```
declare function ddtek:isValid($arg1 as node(), $schema as xs:string) as
xs:boolean external;
```

NOTE: Using this function can degrade performance, depending on the size of the node to be validated and the complexity of the XML schema used to do the validation.

ddtek:javaCast

When possible, Stylus XQuery keeps track of the exact Java class that is associated with a given ddtek:javaObject-typed expression. This is needed to map a given function call to a Java method. When Stylus XQuery is unable to resolve the exact Java class statically, for example when passing ddtek:javaObject as a parameter to or a result from recursive XQuery functions, you can help Stylus XQuery resolve the Java class by specifying the exact class through ddtek:javaCast. See [“Notes About Using Java Instance Methods” on page 326](#) for information about tracking Java classes.

When you use ddtek:javaCast, Stylus XQuery assumes statically that the Java class of the expression that is cast is the class specified in the second argument of the call to the function. At runtime, the object is cast to the specified class. For example:

```
declare namespace A = "ddtekjava:com.ddtek.ejf.A";
```

```
declare namespace B = "ddtekjava:com.ddtek.ejf.B";

declare function A:A() as ddtek:javaObject external;
declare function A:f($this as ddtek:javaObject) as
xs:string external;

declare function local:f($arg1 as ddtek:javaObject, $count
    as xs:integer)
    as ddtek:javaObject {
  if ( $count eq 0 )
  then
    $arg1
  else
    local:f($arg1, $count - 1)
};

let $a := local:f(A:A(), 1)
return A:f(ddtek:javaCast($a, "com.ddtek.qa.ejf.A"))
```

In this example, Stylus XQuery cannot determine the class of the object that is returned by the `local:f` (recursive) function. By using `ddtek:javaCast`, any possible ambiguity is resolved. If, at runtime, the `ddtek:javaCast` operation fails, Stylus XQuery raises an error.

ddtek:ltrim, ddtek:rtrim, and ddtek:trim

These three built-in functions trim whitespaces:

- `ddtek:ltrim` – trims whitespaces to the left of a character string
- `ddtek:rtrim` – trims whitespaces to the right of a character string
- `ddtek:trim` – trims whitespaces to the right and left of a character string

The function declarations for the Stylus XQuery trim functions is:

```
ddtek:ltrim($string as xs:string?) as xs:string?
ddtek:rtrim($string as xs:string?) as xs:string?
ddtek:trim($string as xs:string?) as xs:string?
```

where `$string` is the character string that is to be trimmed.

For example:

```
ddtek:rtrim(" Gustavo ")
```

returns:

```
" Gustavo"
```

and

```
ddtek:trim(" Gustavo ")
```

returns:

```
"Gustavo"
```

ddtek:parse

`ddtek:parse` creates an XQuery Data Model instance from a string value, assuming the string contains well-formed XML. You can use this function to query XML information stored in character columns in database tables. When a database does not support an XML data type, sometimes XML information is stored in character columns. In such cases, `ddtek:parse` allows you to query the XML character data and use it as an XML data source.

The function declaration for `ddtek:parse` is:

```
ddtek:parse($arg as xs:string) as document-node(element(*,xs:untyped))
```

where `$arg` is a well-formed XML document. If `$arg` is not a well-formed XML document, the query execution is aborted and an exception can be handled at the XQJ level.

Assume a database table is created as:

```
create table xmltab (key int primary key, xmlval varchar(2000))
insert into xmltab values (1, '<a><b>1</b><b>11</b></a>')
insert into xmltab values (2, '<a><b>2</b><b>22</b></a>')
```

The following query returns the key value and b elements from the xmlval column for every b element that contains a value greater than 10:

```
for $x in collection('xmltab')/xmltab
let $y := ddtek:parse($x/xmlval)//b[xs:integer(.) > 10]
return (data($x/key), $y)
```

ddtek:serialize

This built-in function controls the process of serializing the query results into XML, XHTML, or HTML notation as specified by XQuery 1.0: An XML Query Language, W3C Recommendation 23 January 2007 located at:

<http://www.w3.org/TR/2007/REC-xquery-20070123/>

The function declaration for ddtek:serialize is:

```
ddtek:serialize($items as item()*, $options as xs:string) as xs:string
```

where:

`$items` specifies the sequence that is to be serialized.

`$options` specifies the serialization options. See [Appendix D “Serialization Support” on page 439](#) for the serialization parameters that you can set using this function.

For example:

```
ddtek:serialize(<books><book/></books>, "indent=yes, omit-xml-declaration=no")
```

returns the xs:string instance:

```
<?xml version="1.0"?>
```

```
<books>
  <book/>
</books>
```

ddtek:serialize-to-url

ddtek:serialize-to-url is equivalent to ddtek:serialize except that instead of returning a string of characters, it writes the result to I/O, allowing you to serialize XML nodes to a specified URI. This function takes advantage of Stylus XQuery *Streaming XML* technique, avoiding the need to load the entire input value into memory. See [“Querying Large XML Documents” on page 177](#) to learn more about Streaming XML.

The function declaration for ddtek:serialize-to-url is:

```
ddtek:serialize-to-url(($items as item()*, $url as
xs:string, $options as xs:string))
```

where:

\$items specifies the input that is to be serialized.

\$url specifies the URI where the result needs to be written. *\$url* must be one of the standard URL schemes supported by Java. If no URL scheme is used, "file:" is implied. If the specified URI references an existing file, the file is overwritten; otherwise it is created.

To specify a ZIP as the target for ddtek:serialize-to-url, prefix the file URI with `zip:.` You can use the `auto-create="yes"` option to specify the creation of a new ZIP file. See [“Working with ZIP Files”](#) for more information.

\$options specifies the serialization options. See [Appendix D “Serialization Support” on page 439](#) for the serialization parameters that you can set using this function.

For example:

```
ddtek:serialize-to-url(<books><book/></books>, file:///c:/result.xml,
"indent=yes, omit-xml-declaration=no")
```

creates a file `c:/result.xml` on the file system with content:

```
<?xml version="1.0"?>
<books>
  <book/>
</books>
```

Working with ZIP Files

In addition to creating new XML documents, you can use `ddtek:serialize-to-url` to create ZIP files, and to create new entries in existing ZIP files.

This example shows how to add a new entry to an existing ZIP file. Note the use of the `zip:` prefix for the file URL:

```
ddtek:serialize-to-url(<e/>, "zip:file:///C:/tmp/test.zip!dir/file.xml", "")
```

This example shows how to add a new entry to a *new* ZIP file, which results from specifying the `auto-create="yes"` option:

```
ddtek:serialize-to-url(<e/>,
"zip:file:///C:/tmp/test.zip!dir/file.xml?auto-create=yes", "")
```

ddtek:sql-delete

The `ddtek:sql-delete` built-in function deletes records in a database table.

The function declaration for `ddtek:sql-delete` is:

```
declare updating function ddtek:sql-delete(
  row as element(*) external;
```

where:

`row` identifies the records to be deleted. Each item in the sequence must be a row element of the database table returned by a previous `fn:collection` call.

The following example deletes all of the records in the holdings database table where the `userid` column equals Minollo.

```
ddtek:sql-delete(collection("holdings")/holdings[userid = "Minollo"])
```

Other examples can be found in the [RDBMSUpdate](#) example.

ddtek:sql-insert

The `ddtek:sql-insert` built-in function inserts a single record in a database table.

The function declaration for `ddtek:sql-insert` is:

```
declare updating function ddtek:sql-insert(
    table as xs:string,
    column as xs:string,
    value as item()*,
    ...) external;
```

where:

`table` is the database table in which to insert the record. The semantics of `table` are equivalent to those for `fn:collection`; see [“Specifying Relational Database Tables” on page 118](#).

`column` is the column of the database table in which to insert a value.

`value` is the value to insert into the specified column.

`column` and `value` are a pair in a variable argument list. If `column` is specified without `value`, an error is raised. You can specify multiple values for this pair, as shown in the example.

The following example inserts a new record with three columns into the holdings table. The columns and their values are `userid=Minollo`, `stockticker=TIVO`, and `shares=200`.

```
ddtek:sql-insert("holdings", "userid", "Minollo", "stockticker", "TIVO",  
  "shares", 200)
```

Other examples can be found in the [RDBMSUpdate](#) example.

ddtek:sql-update

The `ddtek:sql-update` built-in function updates records in a database table.

The function declaration for `ddtek:sql-update` is:

```
declare updating function ddtek:sql-update(  
  row as element()*,  
  column as xs:string,  
  value as item()*,  
  ...) external;
```

where:

`row` identifies the records in the database table to update. Each item in the sequence must be a row element of the database table returned by a previous `fn:collection` call.

`column` is the column of the database table to update.

`value` is the new value for the specified column.

`column` and `value` are a pair in a variable argument list. If `column` is specified without `value`, an error is raised.

The following example updates a record in the holdings table – in particular, the record where the `userid` column equals `Minollo` and the `stockticker` column equals `PRGS`. In this record, the `shares` column is updated to 500.

```
ddtek:sql-update (
  collection("holdings")/holdings[userid="Minollo" and stockticker="PRGS"],
  "shares", 500)
```

Other examples can be found in the [RDBMSUpdate](#) example.

ddtek:validate

`ddtek:validate` allows you to validate an XML element or document node against an element declaration in an XML Schema. Use this function if it is important to validate that:

- An XQuery expression is creating XML results that comply with a given XML Schema declaration
- The XML information provided through an external variable, for example, complies with a given XML Schema before processing it

NOTE: Using this function can degrade performance, depending on the size of the node to be validated and the complexity of the XML Schema used to do the validation.

See the section [ddtek:isValid](#) to learn about alternatives to the `ddtek:validate` function

The function declaration for `ddtek:validate` is:

```
ddtek:validate($arg1 as node(), $arg2 as xs:string) as node() external;
```

where:

`$arg1` is either an XML element or a document node.

`$arg2` is a valid URI. If `$arg2` is a relative URI, it is resolved using the base URI of the XQuery expression.

The function attempts to validate `$arg1` against the XML Schema referred to by `$arg2`. If the validation succeeds, the function

returns `$arg1` unchanged. If the validation fails, Stylus XQuery raises an error.

The following example first validates the input (`req.xml`) against the input XML Schema, `req.xsd`. Then, before returning the results, validates the results against the output XML Schema, `reply.xsd`.

Assume an XML document, `req.xml`, containing the following data:

```
<request-user-holdings>
  <user id="Minollo"/>
  <user id="Jonathan"/>
  <user id="Bill"/>
</request-user-holdings>
```

Also, assume an associated input XML Schema, `req.xsd`:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="request-user-holdings">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="user"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="user">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Lastly, assume a second schema, `reply.xsd`, which is an output XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="reply">
    <xs:complexType>
```

```

    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="user"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="user">
  <xs:complexType>
    <xs:sequence minOccurs="0">
      <xs:element ref="name"/>
      <xs:element maxOccurs="unbounded" ref="holding"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="holding">
  <xs:complexType>
    <xs:attribute name="shares" use="required" type="xs:double"/>
    <xs:attribute name="ticker" use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

The following query returns holding information for each of the users in the input document, req.xml. The query first validates the input against the input XML Schema, req.xsd. Then, before returning the results, the query validates the results against the output XML Schema, reply.xsd.

```

declare base-uri "file:///c:/requests/";
let $validInput := ddtek:validate(doc('req.xml'),'req.xsd')
let $output :=
  <reply>{
    for $req-user in $validInput//user
    let $userid := xs:string($req-user/@id)
    return
      <user id='{ $userid }'>{
        for $db-user in collection('users')/users[userid = $userid]
        return (
          <name>{concat($db-user/firstname, ' ', $db-user/lastname)}</name>,
          for $holding in collection('holdings')/holdings

```

```

    where $holding/userid = $userid
    return
      <holding ticker='{ $holding/stockticker }'
        shares='{ $holding/shares }' />
  )
}</user>
}</reply>
let $validatedOutput := ddtek:validate($output,'reply.xsd')
return $validatedOutput

```

ddtek:validate-and-report

The validate-and-report built-in function provides full error reporting.

The function declaration for ddtek:validate-and-report:

```

declare function ddtek:validate-and-report(
  $arg1 as node(), $schema as xs:string)
as document-node(element(*,xs:untyped))

```

Other built-in functions that provide validation support are [ddtek:isValid](#) and [ddtek:validate](#).

Consider the following example. Given this XML Schema:

```

<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.xquery.com/examples"
  xmlns="http://www.xquery.com/examples">
  <xsd:element name="books">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="title"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

```

        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

This XML document is invalid, because it is missing the `<title>` element:

```

<?xml version="1.0"?>
<p1:books xmlns:p1="http://www.xquery.com/examples"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.xquery.com/examples books.xsd">
  <book>
    <!--<title>My Title</title>-->
  </book>
</p1:books>

```

To capture such validation errors, the `validate-and-report` function can be employed in an XQuery like this one:

```
declare variable $books as document-node(element(*, xs:untyped)) external;
```

```

ddtek:serialize-to-url(
ddtek:validate-and-report($books, "books.xsd"),
"file:///c:/errors/validation-result.xml", "")

```

Which results in the following XML:

```

<p1:validation-result
  xmlns:p1="http://www.datadirect.com/xquery"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.datadirect.com/xquery
    validate-and-report.xsd">
  <error>
    <exception>
      <message>The content of element 'book' is not complete. One of '{title}'
        is expected.</message>
      <stack-trace/>
    </exception>
    <location>

```

```
<publicID/>
<systemID>file:/c:/DDXQ5.0/books-validation.xml</systemID>
<lineNumber>7</lineNumber>
<columnNumber>12</columnNumber>
</location>
</error>
</pl:validation-result>
```

ddtek:wscall

This built-in function allows you to invoke a Web service operation synchronously using SOAP over HTTP. (See <http://www.w3.org/TR/soap11> for details about SOAP.)

The function declarations for ddtek:wscall are:

```
ddtek:wscall($location as element(), $payload as element())
  as document-node( element(*, xs:untyped) )
```

```
ddtek:wscall($location as element(), $header as element(),
  $payload as element())
  as document-node( element(*, xs:untyped) )
```

where:

`$location` is an element named `location` as defined in the following XML Schema:

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.datadirect.com/xquery">
  <xsd:element name="location">
    <xsd:complexType>
      <xsd:attribute name="address" type="xsd:anyURI"/>
      <xsd:attribute name="soapaction" type="xsd:string"
        use="optional"/>
      <xsd:attribute name="timeout" type="xsd:unsignedInt"
        use="optional" default="30000"/>
      <xsd:attribute name="user" type="xsd:string"
        use="optional"/>
      <xsd:attribute name="password" type="xsd:string"
        use="optional"/>
      <xsd:attribute name="http.proxyHost"
        type="xsd:anyURI" use="optional"/>
      <xsd:attribute name="http.proxyPort"
        type="xsd:unsignedInt" use="optional"/>
```



```

<xsd:attribute name="http.proxyUser"
  type="xsd:string" use="optional"/>
<xsd:attribute name="http.proxyPassword"
  type="xsd:string" use="optional"/>
<xsd:attribute name="http.version"
  type="xsd:string" use="optional" default="1.1"/>
<xsd:attribute name="compression"
  type="xsd:string" use="optional" default=""/>
<xsd:attribute name="wrapAnyErrorsAsSOAPFault"
  type="xsd:boolean" use="optional" default="true"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

For example:

```

<ddtek:location
  address="http://www.ejse.com/WeatherService/Service.asmx"
  soapaction="http://ejse.com/WeatherService/GetWeatherInfo"/>

```

The following table defines the attributes in the XML schema.

Attribute Name	Description
address	Web service end point
soapaction	The SOAPAction HTTP header field
timeout	The connection timeout in milliseconds
user	User name for the HTTP connection, if required
password	Password for the HTTP connection if required
http.proxyHost	Proxy host address
http.proxyPort	Proxy port number
http.proxyUser	Proxy user name
http.proxyPassword	Proxy password

Attribute Name	Description
http:version	Version of the HTTP protocol to use when communicating with the server
compression	The type of file compression used to transmit the message. The default is off; you can specify "gzip".
wrapAnyErrorsAsSOAPFault	Determines if exceptions are returned

`$header` is the SOAP Header payload. See the SOAP Header specification at http://www.w3.org/TR/soap11/#_Toc478383497 for details. For example:

```
<tns:UserCreds xmlns:tns="http://ejse.com/WeatherService/">
  <tns:UserName>marypelle@acme.org</tns:UserName>
  <tns:Password>30Mp75Y8p49s</tns:Password>
</tns:UserCreds>
```

`$payload` is an element that defines the payload expected by the web service, which is usually in a format defined by the Web Services Description Language (WSDL). See <http://www.w3.org/TR/wsdl> for details about WSDL. For example:

```
<tns:GetWeatherInfo xmlns:tns="http://ejse.com/WeatherService/">
  <tns:zipCode>01803</tns:zipCode>
</tns:GetWeatherInfo>
```

The return value is a document node that contains the web service response.

The following example invokes a web service that requires registration and uses the SOAP header to specify the user's credentials:

```
ddtek:wscall(
  <ddtek:location
    address="http://www.ejse.com/WeatherService/Service.asmx"
    soapaction="http://ejse.com/WeatherService/GetWeatherInfo"/>,
```

```

<tns:UserCreds xmlns:tns="http://ejse.com/WeatherService/">
  <tns:UserName>marypelle@acme.org</tns:UserName>
  <tns:Password>30Mp75Y8p49s</tns:Password>
</tns:UserCreds>,
<tns:GetWeatherInfo xmlns:tns="http://ejse.com/WeatherService/">
  <tns:zipCode>01803</tns:zipCode>
</tns:GetWeatherInfo>
)

```

HTTP Functions <request> Element

You use attributes of the <request> element to specify optional parameters when submitting a query to the origin server. Examples of <request> element attributes include connection and socket timeout values, the number of retries, and HTTP version.

The following table summarizes the available parameters for HTTP function <request> elements. See [“Specifying HTTP Client-Server Options” on page 244](#) for more information for more information on using the <request> element.

Table 12-8. Function Request Parameters

Parameter Name	Description	Valid Values
cookie-policy	Allows you to specify how to manage cookies. See “Managing Cookies” on page 242 .	RFC2109 (default) RFC2965 netscape ignore-cookies
connection-timeout	Connection timeout value, in milliseconds.	Positive integer
http-version	The HTTP version expected by the origin server.	http_1_0 http_1_1 (default)

Table 12-8. Function Request Parameters

Parameter Name	Description	Valid Values
password	Password associated with “username” when the origin server requires authentication.	String; origin server dependent
protocol-head-body-timeout	The wait time, in milliseconds, for the response body after a ddtek:http-head call. Not all servers adhere to the HTTP protocol when replying to a head request.	Positive integer
protocol-reject-head-body	Reject response returned after ddtek:head-call. Not all servers adhere to the HTTP protocol when replying to a head request.	yes no
proxy-host	URI of a proxy server to which DDXQ can be configured to route all requests.	String
proxy-password	Password associated with the “proxy-username” required by the proxy server.	String; proxy server dependent
proxy-port	Port on the proxy server that is configured to listen for Stylus XQuery requests.	Integer; proxy server dependent
proxy-username	Username associated with the “proxy-password” required by the proxy server.	String; proxy server dependent
response-data-type	Allows you to override the origin server’s default encoding behavior. See “Response Encoding” on page 238 .	text xml base64

Table 12-8. Function Request Parameters

Parameter Name	Description	Valid Values
retries	The number of times Stylus XQuery retries an HTTP call after a failure. Not every failure is subject to a retry.	Positive integer
serialize	Specifies how the payload has to be serialized before sending to the origin server.	String. See “ddtek:serialize” on page 419 for possible values.
socket-linger	The delay, in milliseconds, before a reset is sent to the origin server, allowing more time for data to be read or sent, possibly at the expense of performance.	Positive integer. See “Managing Connections and Sockets” on page 235 .
socket-receivebuffer	Size, in bytes, of the socket receive buffer. The socket receive buffer is an input buffer used by the networking implementation.	Positive integer. See “Managing Connections and Sockets” on page 235 .
socket-sendbuffer	Size, in bytes, of the socket send buffer. The socket send buffer is an output buffer used by the networking implementation.	Positive integer. See “Managing Connections and Sockets” on page 235 .
socket-timeout	The interval, in milliseconds, to wait for data from the origin server.	Positive integer. See “Managing Connections and Sockets” on page 235 .
streaming	Whether or not to allow streaming on the response from the origin server.	yes (default) no

Table 12-8. Function Request Parameters

Parameter Name	Description	Valid Values
tcp-nodelay	Whether to enable or disable an algorithm that trades bandwidth conservation for network latency.	yes (default) no
username	Username associated with “password” when the origin server requires authentication.	String; origin server dependent
wrap-execution	When set to yes, prevents runtime errors from halting query execution.	yes no (default)

Stylus XQuery Options

To learn more about Stylus XQuery options and expression extensions, see [“Using Option Declarations and Extension Expressions” on page 275](#).

Namespaces

This section lists predefined namespaces and namespace prefixes.

Predefined Namespaces (Not Reserved)

[Table C-1](#) lists namespaces that are predefined, but not reserved.

Table C-1. Predefined Namespaces

Namespace Prefix	Namespace URI
ddtek	http://www.datadirect.com/xquery
ddtek-sql	http://www.datadirect.com/xquery/sql-function
ddtek-sql-jdbc	http://www.datadirect.com/xquery/sql-jdbc-escape-function

Predefined Namespaces and Prefixes (Reserved)

The following predefined namespaces are reserved:

- <http://www.datadirect.com/xquery/sql-adaptor>
- <http://www.datadirect.com/xquery/mediator>

The following namespace prefixes are reserved:

- ddtek-fn
- ddtek-me
- ddtek-me-sx
- ddtek-sa
- ddtek-xdt
- ddtek-xs

D Serialization Support

This appendix describes Stylus XQuery support for serialization.

Overview

Serialization is the mechanism that allows you to specify how to generate a query result – as XML or as text, for example. XQuery serialization is described in [XSLT 2.0 and XQuery 1.0 Serialization](#), W3C Recommendation 23 January 2007.

Serialization is an Optional Feature in XQuery. However, XQJ is stricter and requires that every implementation support serialization. XQJ does not require that every parameter defined in the XQuery Serialization specification be supported to its full extent, but it does require that at least a default value for each of the parameters be documented and behave according to the specification.

Serialization Methods

You can serialize query results into a non-XML format using either of the following methods:

- Using the standard XQuery support to serialize query results into XML, XHTML, HTML, and TEXT formats.
- Using the Stylus XML Converters to serialize query results into many other formats, such as Electronic Data

Interchange (EDI) and Comma-Separated Values (CSV). See [Table D-2, “Formats Supported by the Stylus XML Converters,” on page 442](#) for a complete list of the formats supported by the XML Converters.

Using Standard Support

[Table D-1](#) specifies default values for the parameters that control the process of serializing query results into XML, XHTML, HTML, or TEXT notation as specified by the XSLT 2.0 and XQuery 1.0 Serialization, W3C Recommendation 23 January 2007 located at: <http://www.w3.org/TR/2007/REC-xslt-xquery-serialization-20070123/>

Table D-1. Serialization Parameters					
Parameter	XML	XHTML	HTML	TEXT	Modifiable or Hard coded
byte-order-mark	yes	yes	yes	yes	Hard coded
cdata-section-elements	empty	empty	empty	N/A	Modifiable
doctype-public	none	none	none	N/A	Modifiable
doctype-system	none	none	none	N/A	Modifiable
encoding	UTF-8	UTF-8	UTF-8	UTF-8	Modifiable
escape-uri-attributes	N/A	no	no	N/A	Hard coded
include-content-type	N/A	no	no	N/A	Hard coded
indent	no	no	no	N/A	Modifiable, has no effect for HTML
media-type	N/A	empty	empty	empty	Hard coded
method	xml	xhtml	html	text	Modifiable
normalization-form	NFC	NFC	NFC	NFC	Hard coded
omit-xml-declaration	yes	yes	N/A	N/A	Modifiable

Table D-1. Serialization Parameters (cont.)

Parameter	XML	XHTML	HTML	TEXT	Modifiable or Hard coded
standalone	omit	omit	N/A	N/A	Modifiable
undeclare-prefixes	no	no	no	N/A	Hard coded
use-character-maps	empty	empty	empty	empty	Hard coded
version	1.0	1.0	4.01	N/A	Modifiable, has no effect for XML and XHTML

To change a parameter value, use any of the following methods:

- Set the Properties object in the XQJ API.
- Set the serialize option declaration in a query.
- Set the serialize option declaration in DDXQDataSource.
- Use the function, [ddtek:serialize](#).

Example: Setting the Properties Object

```
Properties indentationProperties = new Properties();
indentationProperties.setProperty("indent", "yes");
indentationProperties.setProperty("omit-xml-declaration", "no");
System.out.println(xqSequence.getSequenceAsString(indentationProperties));
```

Example: Setting serialize in a Query

```
declare option ddtek:serialize "indent=yes, omit-xml-declaration=no";
doc("orders.xml")
```

Example: Setting serialize in DDXQDataSource

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
ds.setOptions("serialize=indent=yes,omit-xml-declaration=no");
```

Example: Using ddtek:serialize()

```
ddtek:serialize(<books><book/></books>, "indent=yes, omit-xml-declaration=no")
```

Using the Stylus XML Converters™

To serialize query results using Stylus XML Converters, set the serialization method parameter to a value listed in [Table D-2](#).

Table D-2. Formats Supported by the Stylus XML Converters

Format	Method Parameter Value
Base-64 encoded binary	Base-64
Base-2 to Base-36 encoded binary	Binary
Comma-Separated Values (CSV)	CSV
dBase II, dBase III, dBase III+, dBase IV, and dBase V	dBase_II, dBase_III, dBase_III_plus, dBase_IV, dBase_V
Data Interchange Format (DIF)	DIF
Electronic Data Interchange (EDI)	EDI
E-mail MBox (MBox)	MBox
HTML	HTML
Java .properties	JavaProps
JSON (JavaScript Object Notation)	JSON
Progress OpenEdge .d data dump	DotD
PYX	Pyx
Rich Text Format (RTF)	RTF
Super Data Interchange (SDI)	SDI (Super Data Interchange Format)
Symbolic Link (SYLK)	SYLK (Symbolic Link Format)

Table D-2. Formats Supported by the Stylus XML Converters

Format	Method Parameter Value
Tab-separated values text	TAB (tab-separated values)
Whole-line text	Line
Windows .ini	WinIni
Windows Write	WinWrite

Example: Setting the Properties Object

```

Properties serialization = new Properties();
serialization.setProperty("method", "EDI");
serialization.setProperty("encoding", "UTF-16");
serialization.setProperty("newline", "unix");
xqs.writeSequence(new FileOutputStream("/home/user1/message.x12",
    serialization);

```

Example: Setting serialize in a Query

```

declare option ddtex:serialize "method=EDI,encoding=UTF-16,newline=unix";
doc("orders.xml")

```

Example: Setting serialize in DDXQDataSource

```

DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
ds.setOptions("serialize=method=EDI,encoding=UTF-16,newline=unix");

```

Example: Using ddtex:serialize()

```

ddtex:serialize(
<EANCOM>
  <UNB>
    <UNB01>
      <UNB0101>UNOC</UNB0101>
      <UNB0102>3</UNB0102>
    </UNB01>
    <UNB02>
      <UNB0201>ISENDER</UNB0201>
      <UNB0202>ZZZ</UNB0202>
    </UNB02>
  </UNB>
</EANCOM>

```

```

    </UNB02>
    <UNB03>
      <UNB0301>IRECIPIENT</UNB0301>
      <UNB0302>ZZZ</UNB0302>
    </UNB03>
    <UNB04>
      <UNB0401>080827</UNB0401>
      <UNB0402>1514</UNB0402>
    </UNB04>
    <UNB05>ICONTROL</UNB05>
  </UNB>
  <ORDERS>
    <UNH>
      <UNH01>MESSAGENUMBER</UNH01>
      <UNH02>
        <UNH0201>ORDERS</UNH0201>
        <UNH0202>D</UNH0202>
        <UNH0203>96A</UNH0203>
        <UNH0204>EN</UNH0204>
        <UNH0205>EAN008</UNH0205>
      </UNH02>
    </UNH>
    <BGM/>
    <DTM><DTM01><DTM0101></DTM0101></DTM01></DTM>
    <UNS><UNS01></UNS01></UNS>
    <UNT/>
  </ORDERS>
  <UNZ/>
</EANCOM>
,"method=EDI, encoding=UTF-16, newline=unix")

```

E Database Support

This appendix provides reference information for the databases supported by Stylus XQuery and provides additional information specific to using them.

This appendix contains the following sections:

- [“Supported Databases” on page 445](#)
- [“Data Type Mappings” on page 447](#)
- [“Supported XQuery Atomic Types” on page 459](#)
- [“Database Connection Properties” on page 460](#)

Supported Databases

Stylus XQuery provides support for the following relational databases:

Table 12-9. Stylus XQuery Relational Database Support

Database	Supported Versions
DB2	<ul style="list-style-type: none"> ■ V9.5 for Linux, UNIX, and Windows ■ V9.1 for Linux/UNIX/Windows ■ V9.1 for z/OS (formerly OS/390)
DB2 Universal Database (UDB)	<ul style="list-style-type: none"> ■ v8.2 for Linux/UNIX/Windows ■ v8.2 for Linux/UNIX/Windows ■ v8.1 for z/OS ■ V5R4 for iSeries (formerly AS/400) ■ V5R3 for iSeries (formerly AS/400) ■ V5R2 for iSeries (formerly AS/400)

Table 12-9. Stylus XQuery Relational Database Support

Database	Supported Versions
Informix Dynamic Server	<ul style="list-style-type: none">■ 11■ 10■ 9.4
Microsoft SQL Server	<ul style="list-style-type: none">■ 2008■ 2005■ 2000 Desktop Edition (MSDE 2000)■ 2000 Enterprise Edition (64-bit)
MySQL Enterprise	5.1 and 5.0.x with the following storage engines: <ul style="list-style-type: none">■ InnoDB - Transactional■ MyISAM - Non-Transactional■ Memory (formerly HEAP) - Non-Transactional
Oracle	<ul style="list-style-type: none">■ 11g■ 10g (R1 and R2)■ 9i (R1 and R2)
PostgreSQL	<ul style="list-style-type: none">■ 8.2■ 8.1 <p>NOTE: Requires the PostgreSQL JDBC driver. See “Connection URIs for Third-Party Drivers” on page 143 for information about obtaining and using the driver.</p>
Sybase Adaptive Server Enterprise	<ul style="list-style-type: none">■ 15.0■ 12.5.x

Note: For the most current information, refer to the relational database support table online at:

<https://www.xquery.com/support>

Data Type Mappings

This section describes how database data types are mapped to XML schema data types for supported databases:

- [DB2](#)
- [Informix](#)
- [MySQL](#)
- [Oracle](#)
- [PostgreSQL](#)
- [Microsoft SQL Server](#)
- [Sybase](#)

DB2

[Table E-1](#) describes how DB2 data types are mapped to XML schema data types for Stylus XQuery. Any DB2 data type that is not listed is not supported by Stylus XQuery.

<i>Table E-1. DB2 Data Types</i>	
DB2 Data Type	XML Schema Data Type
Bigint ¹	xs:long
Binary ²	xs:hexBinary
Blob	xs:hexBinary
Char	xs:string
Char for Bit Data	xs:hexBinary
Clob	xs:string
Date	xs:date
DB2XML.XMLClob	xs:anyType
DB2XML.XMLFile	xs:anyType
DB2XML.XMLVarchar	xs:anyType
DBClob	xs:string
Decfloat ³	xs:decimal
Decimal	xs:decimal
Double	xs:double
Double Precision	xs:double
Float	xs:double
Float(n), n > 24	xs:double
Integer	xs:int
Long Varchar for Bit Data	xs:hexBinary
Long Vargraphic	xs:string
Numeric	xs:decimal
Real	xs:float
Rowid ⁴	xs:string

Table E-1. DB2 Data Types *(cont.)*

DB2 Data Type	XML Schema Data Type
Smallint	xs:short
Time	xs:time
Timestamp	xs:dateTime
Varbinary ²	xs:hexBinary
Varchar	xs:string
Varchar for Bit Data	xs:hexBinary
Xml ⁵	xs:anyType

¹ Supported for DB2 for Linux, UNIX, and Windows v8.x and v9.x and for DB2 for z/OS v9.x only.

² Supported for DB2 v9.x for z/OS only.

³ Supported for DB2 for Linux, UNIX, and Windows v9.x and for DB2 for z/OS v9.x only.

⁴ Supported for DB2 for z/OS and DB2 for iSeries only.

⁵ Supported for DB2 for Linux, UNIX, and Windows v9.x only.

Informix

[Table E-2](#) describes how Informix data types are mapped to XML schema data types for Stylus XQuery. Any Informix data type that is not listed is not supported by Stylus XQuery.

Table E-2. Informix Data Types

Informix Data Type	XML Schema Data Type
BLOB	xs:hexBinary
BOOLEAN	xs:boolean
BYTE	xs:hexBinary
CHAR	xs:string

Table E-2. Informix Data Types

Informix Data Type	XML Schema Data Type
CLOB	xs:string
DATE	xs:date
DATETIME HOUR TO SECOND	xs:time
DATETIME YEAR TO DAY	xs:date
DATETIME YEAR TO FRACTION(5)	xs:dateTime
DATETIME YEAR TO SECOND	xs:dateTime
DECIMAL	xs:decimal
FLOAT	xs:double
INT8	xs:long
INTEGER	xs:int
LVARCHAR	xs:string
MONEY	xs:decimal
NCHAR	xs:string
NVARCHAR	xs:string
SERIAL	xs:int
SERIAL8	xs:long
SMALLFLOAT	xs:float
SMALLINT	xs:short
TEXT	xs:string
VARCHAR	xs:string

MySQL

[Table E-3](#) describes how MySQL data types are mapped to XML schema data types for Stylus XQuery. Any MySQL data type that is not listed is not supported by Stylus XQuery.

Table E-3. MySQL Enterprise Data Types

MySQL Enterprise Data Type	XML Schema Data Type
BIGINT	xs:long
BIGINT UNSIGNED	xs:decimal
BINARY	xs:hexBinary
BIT	xs:hexBinary
BLOB	xs:hexBinary
BOOLEAN	xs:boolean
DATE	xs:date
DATETIME	xs:dateTime
DECIMAL	xs:decimal
DECIMAL UNSIGNED	xs:decimal
DOUBLE	xs:double
DOUBLE UNSIGNED	xs:double
FLOAT	xs:float
FLOAT UNSIGNED	xs:float
INTEGER	xs:int
INTEGER UNSIGNED	xs:long
LOBLOB	xs:hexBinary
LONGTEXT	xs:string
MEDIUMBLOB	xs:hexBinary
MEDIUMINT	xs:int
MEDIUMINT UNSIGNED	xs:int
MEDIUMTEXT	xs:string
SMALLINT	xs:short
SMALLINT UNSIGNED	xs:int

Table E-3. MySQL Enterprise Data Types

MySQL Enterprise Data Type	XML Schema Data Type
TEXT	xs:string
TIME	xs:time
TIMESTAMP	xs:dateTime
TINYBLOB	xs:hexBinary
TINYINT	xs:short
TINYINT UNSIGNED	xs:short
TINYTEXT	xs:string
VARBINARY	xs:hexBinary
VARCHAR	xs:string
YEAR	xs:short

Oracle

[Table E-4](#) describes how Oracle data types are mapped to XML schema data types for Stylus XQuery. Any Oracle data type that is not listed is not supported by Stylus XQuery.

Table E-4. Oracle Data Types

Oracle Data Type	XML Schema Data Type
BFILE	xs:hexBinary
BINARY_FLOAT	xs:float
BINARY_DOUBLE	xs:double
BLOB	xs:hexBinary
CHAR	xs:string
CLOB	xs:string
DATE	xs:dateTime

Table E-4. Oracle Data Types *(cont.)*

Oracle Data Type	XML Schema Data Type
FLOAT(n)	xs:double
LONG	xs:string
LONG RAW	xs:hexBinary
NCHAR	xs:string
NCLOB	xs:string
NUMBER (p, s)	xs:decimal
NUMERIC	xs:decimal
NVARCHAR2	xs:string
RAW	xs:hexBinary
REAL	xs:double
ROWID	xs:string
SMALLINT	xs:decimal
TIMESTAMP	xs:dateTime
TIMESTAMP WITH LOCAL TIME ZONE	xs:dateTime
TIMESTAMP WITH TIME ZONE	xs:dateTime
UROWID	xs:string
VARCHAR2	xs:string
XMLTYPE ¹	xs:anyType

¹ Supported for Oracle 10gR2 and 11gR1 only.

PostgreSQL

Table E-5 describes how PostgreSQL data types are mapped to XML schema data types for Stylus XQuery. Any PostgreSQL data type that is not listed is not supported by Stylus XQuery.

Table E-5. PostgreSQL Data Types	
PostgreSQL Data Type	XML Schema Data Type
bigserial	xs:long
bit	xs:boolean
bool	xs:boolean
bytea	xs:hexBinary
char	xs:string
date	xs:date
float4	xs:double
float8	xs:float
int2	xs:short
int4	xs:int
int8	xs:long
money	xs:float
name	xs:string
numeric	xs:decimal
oid	xs:int
serial	xs:int
text	xs:string
time	xs:time
timestamp	xs:dateTime
timestamp with time zone	xs:dateTime
time with time zone	xs:time
varchar	xs:string

Microsoft SQL Server

[Table E-6](#) describes how Microsoft SQL Server data types are mapped to XML schema data types for Stylus XQuery. Any Microsoft SQL Server data type that is not listed is not supported by Stylus XQuery.

Table E-6. Microsoft SQL Server Data Types

Microsoft SQL Server Data Type	XML Schema Data Type
bigint	xs:long
bigint identity	xs:long
binary	xs:hexBinary
bit	xs:boolean
char	xs:string
datetime	xs:dateTime
decimal	xs:decimal
decimal() identity	xs:decimal
double	xs:double
float	xs:double
image	xs:hexBinary
int	xs:int
int identity	xs:int
money	xs:decimal
nchar	xs:string
ntext	xs:string
numeric	xs:decimal
numeric() identity	xs:decimal
nvarchar	xs:string
real	xs:float
rowversion	xs:hexBinary
sql_variant	xs:string

¹Supported for Microsoft SQL Server 2005 and 2008 only.

Table E-6. Microsoft SQL Server Data Types *(cont.)*

Microsoft SQL Server Data Type	XML Schema Data Type
smalldatetime	xs:dateTime
smallint	xs:short
smallint identity	xs:short
smallmoney	xs:decimal
sysname	xs:string
text	xs:string
timestamp	xs:hexBinary
tinyint	xs:short
tinyint identity	xs:short
uniqueidentifier	xs:string
varbinary	xs:hexBinary
varchar	xs:string
xml ¹	xs:anyType

¹Supported for Microsoft SQL Server 2005 and 2008 only.

Sybase

[Table E-7](#) describes how Sybase data types are mapped to XML schema data types for Stylus XQuery. Any Sybase data type that is not listed is not supported by Stylus XQuery.

Table E-7. Sybase Data Types

Sybase Data Type	XML Schema Data Type
BIGINT ¹	xs:long
BINARY(n)	xs:hexBinary
BIT	xs:boolean
CHAR(n)	xs:string
CHAR VARYING	xs:string
CHARACTER	xs:string
CHARACTER VARYING	xs:string
DATE	xs:date
DATETIME	xs:dateTime
DEC	xs:decimal
DECIMAL(p,s)	xs:decimal
DOUBLE	xs:double
FLOAT	xs:double
IMAGE	xs:hexBinary
INT	xs:int
INTEGER	xs:int
MONEY	xs:decimal
NATIONAL CHAR	xs:string
NATIONAL CHAR VARYING	xs:string
NCHAR(n)	xs:string
NUMERIC(p,s)	xs:decimal
NVARCHAR(n)	xs:string
REAL	xs:float

¹Supported only for Sybase 15.0.

Table E-7. Sybase Data Types *(cont.)*

Sybase Data Type	XML Schema Data Type
SMALLDATETIME	xs:dateTime
SMALLINT	xs:short
SMALLMONEY	xs:decimal
TEXT	xs:string
TIME	xs:time
TINYINT	xs:short
UNICHAR	xs:string
UNICODE CHAR VARYING	xs:string
UNICODE CHARACTER	xs:string
UNICODE CHARACTER VARYING	xs:string
UNIVARCHAR	xs:string
VARBINARY(n)	xs:hexBinary
VARCHAR(n)	xs:string

¹Supported only for Sybase 15.0.

Supported XQuery Atomic Types

[Table E-8](#) lists the XQuery atomic types that are supported by Stylus XQuery for both XML and relational sources.

Table E-8. Predefined XQuery Atomic Types

xs:untypedAtomic	xs:positiveInteger
xs:anyAtomicType	xs:base64Binary
xs:string	xs:hexBinary
xs:boolean	xs:duration
xs:decimal	xs:yearMonthDuration
xs:float	xs:dayTimeDuration
xs:double	xs:QName
xs:dateTime	xs:anyURI
xs:date	xs:gDay
xs:time	xs:gMonth
xs:integer	xs:gMonthDay
xs:long	xs:gYear
xs:int	xs:gYearMonth
xs:short	xs:normalizedString
xs:byte	xs:token
xs:nonPositiveInteger	xs:language
xs:negativeInteger	xs:NMTOKEN
xs:nonNegativeInteger	xs:Name
xs:unsignedLong	xs:NCName
xs:unsignedInt	xs:ID
xs:unsignedShort	xs:IDREF
xs:unsignedByte	xs:ENTITY

Database Connection Properties

This section describes the connection properties you can specify for databases supported by Stylus XQuery.

See [“Specifying Connection URIs” on page 141](#) for information about the format of connection URLs and specifying connection properties in connection URLs.

This section provides information for the following supported databases:

- [“DB2” on page 460](#)
- [“Informix” on page 466](#)
- [“Microsoft SQL Server” on page 467](#)
- [“MySQL Enterprise” on page 470](#)
- [“Oracle” on page 471](#)
- [“PostgreSQL” on page 480](#)
- [“Sybase” on page 481](#)

DB2

This section describes the connection properties you can specify for DB2 and provides information about how Stylus XQuery creates DB2 packages.

[Table E-9](#) lists the connection properties supported for DB2 by Stylus XQuery.

Table E-9. DB2 Connection Properties

DB2 Property	Description
AuthenticationMethod	<p>{kerberos encryptedUIDPassword encryptedPassword clearText client}. Determines which authentication method Stylus XQuery uses when establishing a connection.</p> <p>If set to kerberos, Stylus XQuery uses Kerberos authentication. Stylus XQuery ignores any user ID or password specified. See “Using Kerberos Authentication” on page 147 for more information about using Kerberos.</p> <p>If set to encryptedUIDPassword, Stylus XQuery uses user ID/password authentication. Stylus XQuery sends an encrypted user ID and password to the DB2 server for authentication. If a user ID and password are not specified, an exception is thrown.</p> <p>If set to encryptedPassword, Stylus XQuery uses user ID/password authentication. Stylus XQuery sends a user ID in clear text and an encrypted password to the DB2 server for authentication. If a user ID and password are not specified, an exception is thrown.</p> <p>If set to clearText (the default), Stylus XQuery uses user ID/password authentication. Stylus XQuery sends the user ID and password in clear text to the DB2 server for authentication. If a user ID and password are not specified, an exception is thrown.</p> <p>If set to client, Stylus XQuery uses client authentication. The DB2 server relies on the client to authenticate the user and does not provide additional authentication. Stylus XQuery ignores any user ID or password specified.</p> <p>The User property provides the user ID. The Password property provides the password.</p> <p>If the specified authentication method is not supported by the DB2 server, the connection fails and an exception is thrown.</p>

Table E-9. DB2 Connection Properties *(cont.)*

DB2 Property	Description
CreateDefaultPackage	<p>{true false}. If set to true, the required DB2 packages are automatically created even if they already exist. Existing DB2 packages are replaced by the new packages. If set to false (the default), the required DB2 packages are created automatically only if they do not already exist.</p> <p>For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the ReplacePackage property.</p> <p>For DB2 for z/OS and DB2 for iSeries, DB2 packages are created in the collection or library specified by the PackageCollection property.</p> <p>See “Creating DB2 Packages” on page 464 for more information about creating DB2 packages.</p>
DatabaseName Required for Linux/UNIX/Windows	<p>The name (DB2 for Linux/UNIX/Windows) or location name (DB2 for z/OS and DB2 for iSeries) of the database you want to connect to.</p>
DynamicSections	<p>The maximum number of prepared statements that can be open at any time. This value must be a positive integer. The default is 200.</p>
Grantee	<p>The name of the schema to which you want to grant EXECUTE privileges for DB2 packages. The value must be a valid DB2 schema. This property is ignored if the GrantExecute property is set to false.</p> <p>The default is PUBLIC.</p> <p>See “Creating DB2 Packages” on page 464 for more information about creating DB2 packages.</p>
GrantExecute	<p>{true false}. Determines which DB2 schema is granted EXECUTE privileges for DB2 packages. If set to true (the default), EXECUTE privileges are granted to the schema specified by the Grantee property. If set to false, EXECUTE privileges are granted to the schema that created the DB2 packages.</p> <p>See “Creating DB2 Packages” on page 464 for more information about creating DB2 packages.</p>

Table E-9. DB2 Connection Properties (cont.)

DB2 Property	Description
InitializationString	<p>Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection.</p> <p>Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. The following example adds USER2 to the CURRENT PATH special register and sets the CURRENT PRECISION special register to DEC31.</p> <pre>jdbc:xquery:db2://server1:50000; InitializationString=(SET CURRENT PATH=current_path, USER2;SET CURRENT PRECISION='DEC31')</pre> <p>NOTE: Setting the CURRENT PRECISION special register is only valid for DB2 for z/OS.</p> <p>If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an exception indicating which SQL command or commands failed.</p>
LocationName Required for z/OS and iSeries if DatabaseName is not specified	<p>The name of the DB2 location that you want to access.</p> <p>For DB2 for z/OS, your system administrator can determine the name of your DB2 location using the following command:</p> <pre>DISPLAY DDF</pre> <p>For DB2 for iSeries, your system administrator can determine the name of your DB2 location using the following command. The name of the database that is listed as *LOCAL is the value you should use for this property.</p> <pre>WRKRDBDIRE</pre> <p>This property is supported only for DB2 for z/OS and DB2 for iSeries.</p>
PackageCollection	<p>The name of the collection or library (group of packages) to which DB2 packages are bound. The default is NULLID.</p> <p>This property is ignored for DB2 for Linux/UNIX/Windows.</p>

Table E-9. DB2 Connection Properties *(cont.)*

DB2 Property	Description
PackageOwner	The owner to be used for any DB2 packages that are created. The default is null. See “Creating DB2 Packages” on page 464 for more information about creating DB2 packages.
Password	A case-sensitive password used to connect to your DB2 database. A password is required only if user ID/password authentication is enabled on your database. Contact your system administrator to obtain your password.
ReplacePackage	{true false}. Specifies whether the current bind process will replace the existing DB2 packages. If set to true, the current bind process will replace the existing DB2 packages. If set to false (the default), the current bind process will not replace the existing DB2 packages. For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the CreateDefaultPackage property. See “Creating DB2 Packages” on page 464 for more information about creating DB2 packages.
SecurityMechanism DEPRECATED	This property is recognized for backward compatibility, but we recommend that you use the AuthenticationMethod property to set the authentication method used by Stylus XQuery.
User	The case-sensitive user name used to connect to the DB2 database.

Creating DB2 Packages

Stylus XQuery automatically creates all required DB2 packages at connection time, which, by default, contain 200 dynamic sections and are created in the NULLID collection (or library). You can override the default number of dynamic sections by setting the DynamicSections property. Similarly, you can override the collection in which packages are created by setting the PackageCollection property.

NOTE: The initial connection may take a few minutes because of the number and size of the packages that must be created for the connection. Subsequent connections do not incur this delay.

In most cases, you do not need to create DB2 packages because they are automatically created at connection time. If, for some reason, you need to explicitly create them, you can create them using the following sets of connection properties.

NOTE: The user ID creating the DB2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

For DB2 for Linux/UNIX/Windows

- CreateDefaultPackage=true
- ReplacePackage=true
- DynamicSections=x (where x is a positive integer)

NOTE: To create new DB2 packages, you must use ReplacePackage=true in conjunction with CreateDefaultPackage=true. If a DB2 package already exists, it is replaced when ReplacePackage=true.

The following URL creates DB2 packages with 400 dynamic sections. If any DB2 packages already exist, they are replaced by the new ones being created.

```
jdbc:xquery:db2://server1:50000;databaseName=SAMPLE;  
createDefaultPackage=TRUE;replacePackage=TRUE;dynamicSections=400
```

For DB2 for z/OS and iSeries

- PackageCollection=*collection_name* (where *collection_name* is the name of the collection or library to which DB2 packages are bound)
- CreateDefaultPackage=true
- DynamicSections=x (where x is a positive integer)

The following URL creates DB2 packages with 400 dynamic sections.

```
jdbc:xquery:db2://server1:50000;locationName=SAMPLE;  
packageCollection=DEFAULT;createDefaultPackage=TRUE;dynamicSections=400
```

Informix

This section describes the connection properties you can specify for Informix.

[Table E-10](#) lists the connection properties supported for Informix by Stylus XQuery.

Table E-10. Informix Connection Properties

Informix Property	Description
DatabaseName	The name of the database you want to which you want to connect. If this property is not specified, a connection is established to the specified server without connecting to a particular database.
InformixServer Required	The name of the Informix database server to which you want to connect.
InitializationString	Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection. For example: <code>InitializationString=command</code> Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. For example: <code>jdbc:xquery:informix://server1:2003;informixServer=test_server;databaseName=test;initializationString=(command1;command2)</code> If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an exception indicating which SQL command or commands failed.

Table E-10. Informix Connection Properties (cont.)

Informix Property	Description
Password	A case-insensitive password used to connect to your Informix database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password.
User	The case-insensitive default user name used to connect to your Informix database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name.

Microsoft SQL Server

This section describes the connection properties you can specify for Microsoft SQL Server.

[Table E-11](#) lists the connection properties supported for Microsoft SQL Server by Stylus XQuery.

Table E-11. Microsoft SQL Server Connection Properties

SQL Server Property	Description
AuthenticationMethod	<p>{auto kerberos ntlm userIdPassword}. Determines which authentication method Stylus XQuery uses when establishing a connection.</p> <p>If set to auto (the default), Stylus XQuery uses SQL Server authentication, Kerberos authentication, or NTLM authentication when establishing a connection. Stylus XQuery selects an authentication method based on a combination of criteria such as whether the application provides a user ID, Stylus XQuery is running on a Windows platform, and Stylus XQuery can load the DLL required for NTLM authentication.</p> <p>If set to kerberos, Stylus XQuery uses Kerberos authentication. Stylus XQuery ignores any user ID or password specified. See “Using Kerberos Authentication” on page 147 for more information about using Kerberos.</p> <p>If set to ntlm, Stylus XQuery uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If Stylus XQuery cannot load the DLL, Stylus XQuery raises an error. Stylus XQuery ignores any user ID or password specified. See “Using NTLM Authentication” on page 159 for more information about NTLM authentication.</p> <p>If set to userIdPassword, Stylus XQuery uses SQL Server authentication when establishing a connection. If a user ID is not specified, an exception is thrown.</p> <p>The “User” property provides the user ID. The “Password” property provides the password.</p>
DatabaseName	The name of the database to which you want to connect.

Table E-11. Microsoft SQL Server Connection Properties (cont.)

SQL Server Property	Description
InitializationString	<p>Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection. For example:</p> <pre>InitializationString=command</pre> <p>Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. For example:</p> <pre>jdbc:xquery:sqlserver://server1:1433;databaseName=test;initializationString=(command1;command2)</pre> <p>If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an exception indicating which SQL command or commands failed.</p>
LoadLibraryPath	<p>Specifies the directory Stylus XQuery looks in for the DLL used for NTLM authentication. The value is the fully qualified path of the directory that contains the DLL. When you install Stylus XQuery, the NTLM DLLs are placed in the <i>install_dir/lib</i> subdirectory, where <i>install_dir</i> is the Stylus XQuery installation directory.</p> <p>By default, Stylus XQuery looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install Stylus XQuery in a directory that is not on the Windows system path, you can set this property to specify the location of the NTLM authentication DLLs. For example, if you install Stylus XQuery in a directory named "StylusXQuery" that is not on the Windows system path, you can use this property to specify the directory containing the NTLM authentication DLLs.</p> <pre>jdbc:xquery:sqlserver://server3:1433; databaseName=test;loadLibraryPath=C:\StylusXQuery\lib; User=test;Password=secret</pre> <p>See "Using NTLM Authentication" on page 159 for more information about NTLM authentication.</p>

Table E-11. Microsoft SQL Server Connection Properties *(cont.)*

SQL Server Property	Description
Password	A case-insensitive password used to connect to your Microsoft SQL Server database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password.
User	The case-insensitive user name used to connect to your Microsoft SQL Server database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name.

MySQL Enterprise

This section describes the connection properties you can specify for MySQL Enterprise.

NOTE: You must purchase commercially licensed MySQL database software or a MySQL Enterprise subscription in order to use Stylus XQuery with MySQL software.

[Table E-12](#) lists the connection properties supported for MySQL Enterprise by Stylus XQuery.

Table E-12. MySQL Enterprise Connection Properties

MySQL Enterprise Property	Description
DatabaseName	The name of the database you want to connect to.

Table E-12. MySQL Enterprise Connection Properties (cont.)

MySQL Enterprise Property	Description
InitializationString	<p>Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection. For example:</p> <pre>InitializationString=command</pre> <p>Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. For example:</p> <pre>jdbc:xquery:mysql://server1:3306;databaseName=test;initializationString=(command1;command2)</pre> <p>If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an exception indicating which SQL command or commands failed.</p>
Password	<p>A case-insensitive password used to connect to your MySQL Enterprise database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password.</p>
User	<p>The case-insensitive default user name used to connect to your MySQL Enterprise database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name.</p>

Oracle

This section describes the connection properties you can specify for Oracle and provides information about using tnsnames.ora files to specify connection information to an Oracle database.

[Table E-13](#) lists the connection properties supported for Oracle by Stylus XQuery.

Table E-13. Oracle Connection Properties

Oracle Property	Description
AuthenticationMethod	<p>{auto kerberos kerberosUIDPassword ntlm client userIDPassword}. Determines which authentication method Stylus XQuery uses when establishing a connection.</p> <p>If set to auto (the default), Stylus XQuery uses user ID/password, Kerberos, or NTLM authentication when establishing a connection. Stylus XQuery selects an authentication method based on a combination of criteria, such as whether the application provides a user ID, Stylus XQuery is running on a Windows platform, and Stylus XQuery can load the DLL required for NTLM authentication.</p> <p>If set to kerberos, Stylus XQuery uses Kerberos authentication. Stylus XQuery ignores any user ID or password specified.</p> <p>If set to kerberosUIDPassword, Stylus XQuery first uses Kerberos to authenticate the user. Next, Stylus XQuery reauthenticates the user using user ID/password authentication. If a user ID and password are not specified, Stylus XQuery throws an exception. If either Kerberos or user ID/password authentication fails, the connection attempt fails and Stylus XQuery throws an exception.</p> <p>If set to ntlm, Stylus XQuery uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If Stylus XQuery cannot load the DLL, it throws an exception. Stylus XQuery ignores any user ID or password specified. This value is supported for Windows clients only. See “Using NTLM Authentication” on page 159 for more information about NTLM authentication.</p> <p>If set to client, Stylus XQuery uses client authentication. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication. Stylus XQuery ignores any user ID or password specified.</p> <p>If set to userIDPassword, Stylus XQuery uses user ID/password authentication. If a user ID and password are not specified, an exception is thrown.</p> <p>The User property provides the user ID. The Password property provides the password.</p>

Table E-13. Oracle Connection Properties (cont.)

Oracle Property	Description
InitializationString	<p>Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection. For example:</p> <pre>initializationString=command</pre> <p>Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. For example:</p> <pre>jdbc:xquery:oracle://server1:1521;ServiceName=ORCL;iInitializationString=(command1;command2)</pre> <p>If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an error indicating which SQL command or commands failed.</p>
LoadLibraryPath	<p>Specifies the directory Stylus XQuery looks in for the DLL used for NTLM authentication. The value is the fully qualified path of the directory that contains the DLL. When you install Stylus XQuery, the NTLM DLLs are placed in the <i>install_dir/lib</i> subdirectory, where <i>install_dir</i> is the Stylus XQuery installation directory.</p> <p>By default, Stylus XQuery looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install Stylus XQuery in a directory that is not on the Windows system path, you can set this property to specify the location of the NTLM authentication DLLs. For example, if you install Stylus XQuery in a directory named "StylusXQuery" that is not on the Windows system path, you can use this property to specify the directory containing the NTLM authentication DLLs.</p> <pre>jdbc:xquery:oracle://server3:1521;serviceName=ORCL;loadLibraryPath=C:\StylusXQuery\lib;User=test;Password=secret</pre> <p>See “Using NTLM Authentication” on page 159 for more information about NTLM authentication.</p>

Table E-13. Oracle Connection Properties *(cont.)*

Oracle Property	Description
Password	A case-insensitive password used to connect to your Oracle database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password.
ServiceName	<p>The database service name that specifies the database used for the connection. This property is mutually exclusive with the SID property. The service name is a string that is the global database name—a name that typically comprises the database name and domain name. For example:</p> <p><code>sales.us.acme.com</code></p> <p>This property is useful to specify connections to an Oracle Real Application Clusters (RAC) system rather than a specific Oracle instance because the nodes in a RAC system share a common service name.</p> <p>If using a <code>tnsnames.ora</code> file to provide connection information, do not specify this property.</p> <p>See “Using Oracle tnsnames.ora Files” on page 476 for information about specifying the database service name using a <code>tnsnames.ora</code> file.</p>
SID	<p>The Oracle System Identifier that refers to the instance of the Oracle database running on the server.</p> <p>The default is <code>ORCL</code>, which is the default SID that is configured when installing your Oracle database.</p> <p>If using a <code>tnsnames.ora</code> file to provide connection information, do not specify this property.</p> <p>See “Using Oracle tnsnames.ora Files” on page 476 for information about specifying an Oracle SID using a <code>tnsnames.ora</code> file.</p>

Table E-13. Oracle Connection Properties (cont.)

Oracle Property	Description
TNSNamesFile	<p>The path and filename to the tnsnames.ora file from which connection information is retrieved. The tnsnames.ora file contains connection information that is mapped to Oracle net service names. Using a tnsnames.ora file to centralize connection information simplifies maintenance when changes occur.</p> <p>The value of this property must be a valid path and filename to a tnsnames.ora file.</p> <p>If you specify this property is specified, then:</p> <ul style="list-style-type: none"> ■ Also specify the TNSServerName property. ■ Do not specify any of following: PortNumber (in URL), ServerName (in URL), or SID (connection property). <p>If any of this connection information is specified in addition to this property, an error is generated. See “Using Oracle tnsnames.ora Files” on page 476 for information about using tnsnames.ora files to connect.</p>
TNSServerName	<p>The Oracle net service name used to reference the connection information in a tnsnames.ora file. The value of this property must be a valid net service name entry in the tnsnames.ora file specified by the TNSNamesFile property.</p> <p>If this property is specified, you also must specify the TNSNamesFile property.</p> <p>If this property is specified, do not specify any of the following connection information to prevent conflicts:</p> <p>PortNumber (in URL) ServerName (in URL) SID (connection property)</p> <p>If any of this connection information is specified in addition to this property, an error is generated. See “Using Oracle tnsnames.ora Files” on page 476 for information about using tnsnames.ora files to connect.</p>
User	<p>The case-insensitive user name used to connect to your Oracle database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name.</p>

Using Oracle *tnsnames.ora* Files

The *tnsnames.ora* file is used to map connection information for each Oracle service to a logical alias. Stylus XQuery can retrieve basic connection information from a *tnsnames.ora* file, including:

- Oracle server name and port
- Oracle System Identifier (SID) or Oracle service name

In a *tnsnames.ora* file, connection information for an Oracle service is associated with an alias, or Oracle net service name. Each net service name entry contains connect descriptors that define listener and service information. The following example in [Figure E-1](#) shows connection information in a *tnsnames.ora* file configured for the net service name entry, FITZGERALD.SALES.

Figure E-1. *tnsnames.ora* Example

```
FITZGERALD.SALES =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP) (HOST = server1) (PORT = 1521))
    (CONNECT_DATA =
      (SID = ORCL)
    )
  )
```

Using this example, if the Oracle net service name entry FITZGERALD.SALES is referenced, Stylus XQuery would connect to the Oracle database instance identified by the Oracle SID ORCL (*SID=ORCL*).

Typically, a *tnsnames.ora* file is installed when you install an Oracle database. By default, the *tnsnames.ora* file is located in the ORACLE_HOME\network\admin directory on Windows and the \$ORACLE_HOME/network/admin directory on UNIX and Linux.

Connecting to the Database

To retrieve connection information from an Oracle `tnsnames.ora` file, you must inform Stylus XQuery which `tnsnames.ora` file (using the `TNSNamesFile` property) and Oracle service name entry (using the `TNSServerName` property) to use so that the correct connection information is referenced. For example, the following connection URL:

```
jdbc:xquery:oracle:TNSNamesFile=c:\oracle92\NETWORK\ADMIN\tnsnames.ora;
TNSServerName=FITZGERALD.SALES
```

specifies the path and filename of the `tnsnames.ora` file (`TNSNamesFile=c:\oracle92\NETWORK\ADMIN\tnsnames.ora`) and the net service name entry (`TNSServerName=FITZGERALD.SALES`) to use for the connection.

NOTE: The connection URL does not specify the server name and port of the database server; that information is specified in the `tnsnames.ora` file referenced by the `TNSNamesFile` property.

If using `tnsnames.ora` files with a Security Manager on a Java 2 Platform, read permission must be granted to the `tnsnames.ora` file. For example:

```
grant codeBase "file:/install_dir/lib/-" {
    permission java.io.FilePermission "C:\\oracle\\ora92\\network\\admin\\
        tnsnames.ora", "read";
};
```

where `install_dir` is the Stylus XQuery installation directory.

Configuring the *tnsnames.ora* File

If using a *tnsnames.ora* file to retrieve connection information, do not specify any of the following connection information in a connection URL or property to prevent conflicts:

- PortNumber (in URL)
- ServerName (in URL)
- ServiceName (connection property)
- SID (connection property)

[Table E-14](#) lists the Oracle connection properties that correspond to *tnsnames.ora* connect descriptor parameters.

Table E-14. Oracle Connection Property Mappings to *tnsnames.ora* Connect Descriptor Parameters

Oracle Connection Property	<i>tnsnames.ora</i> Parameter
PortNumber = <i>port</i>	<div>PORT = <i>port</i></div> <div>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The PORT parameter is used within the ADDRESS parameter to specify the port number for each server entry. For example:</div> <div>(ADDRESS_LIST= (ADDRESS= (PROTOCOL = TCP) (HOST = server1) (PORT = 1521)) ...)</div> <div>A port of 1521, the default port number when installing an Oracle database, is specified for server1.</div>

Table E-14. Oracle Connection Property Mappings to tnsnames.ora Connect Descriptor Parameters *(cont.)*

Oracle Connection Property	tnsnames.ora Parameter
ServerName = <i>server_name</i>	<p>HOST = <i>server_name</i></p> <p>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The HOST parameter is used within the ADDRESS parameter to specify the server name for each server entry. The server entry can be an IP address or a server name. For example:</p> <pre>(ADDRESS_LIST= (ADDRESS= (PROTOCOL = TCP) (HOST = server1) (PORT = 1521)) ...)</pre> <p>The server name server1 is specified in the first server entry.</p>
ServiceName = <i>service_name</i>	<p>SERVICE_NAME = <i>service_name</i></p> <p>The database service name that specifies the database used for the connection. The service name is a string that is the global database name—a name that typically comprises the database name and domain name. For example:</p> <pre>sales.us.acme.com</pre> <p>The service name is specified in the CONNECT_DATA parameter. For example:</p> <pre>(CONNECT_DATA= (SERVICE_NAME=sales.us.acme.com))</pre> <p>This parameter is mutually exclusive with the SID attribute and is useful to specify connections to an Oracle Real Application Clusters (RAC) system rather than a specific Oracle instance.</p>

Table E-14. Oracle Connection Property Mappings to tnsnames.ora Connect Descriptor Parameters *(cont.)*

Oracle Connection Property	tnsnames.ora Parameter
SID = <i>SID</i>	<div>SID = <i>SID</i></div> <div>The Oracle System Identifier (SID) that refers to the instance of the Oracle database running on the server. The default Oracle SID that is configured when installing your Oracle database software is ORCL. The SID is specified in the CONNECT_DATA parameter. For example:</div> <div>(CONNECT_DATA= (SID=ORCL))</div> <div>This parameter is mutually exclusive with the SERVICE_NAME attribute.</div>

For more information about configuring tnsnames.ora files, refer to your Oracle documentation.

PostgreSQL

Refer to your PostgreSQL JDBC driver documentation for information about connection properties supported by the PostgreSQL JDBC driver.

Sybase

This section describes the connection properties you can specify for Sybase.

[Table E-15](#) lists the connection properties supported for Sybase by Stylus XQuery.

Table E-15. Sybase Connection Properties

Sybase Property	Description
AuthenticationMethod	<p>{kerberos userIdPassword}. Determines which authentication method Stylus XQuery uses when establishing a connection.</p> <p>If set to kerberos, Stylus XQuery uses Kerberos authentication. Stylus XQuery ignores any user ID or password specified. If you set this value, you also must set the ServicePrincipalName property. See “Using Kerberos Authentication” on page 147 for more information about using Kerberos.</p> <p>If set to userIdPassword (the default), Stylus XQuery uses user ID/password authentication. If a user ID and password are not specified, an exception is thrown.</p> <p>The User property provides the user ID. The Password property provides the password.</p>
InitializationString	<p>Specifies one or multiple SQL commands to be executed by Stylus XQuery after it has established the connection to the database and has performed all initialization for the connection. For example:</p> <pre>InitializationString=command</pre> <p>Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. For example:</p> <pre>jdbc:xquery:sybase://server1:5000;DatabaseName=test;InitializationString=(command1;command2)</pre> <p>If the execution of a SQL command fails, the connection attempt also fails and Stylus XQuery raises an exception indicating which SQL command or commands failed.</p>

Table E-15. Sybase Connection Properties *(cont.)*

Sybase Property	Description
Password	A case-insensitive password used to connect to your Sybase database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password.
ServicePrincipalName	<p>Specifies the case-sensitive service principal name to be used by Stylus XQuery for Kerberos authentication. For Sybase, the service principal name is the name of a server configured in your Sybase interfaces file. If you set this property, you also must set the value of the “AuthenticationMethod” property to Kerberos.</p> <p>The value of this property can include the Kerberos realm name, but it is optional. If you do not specify the Kerberos realm name, the default Kerberos realm is used. For example, if the service principal name, including Kerberos realm name, is <code>server/sybase125ase1@XYZ.COM</code> and the default realm is <code>XYZ.COM</code>, valid values for this property are:</p> <p><code>server/sybase125ase1@XYZ.COM</code> and <code>server/sybase125ase1</code></p> <p>When Kerberos authentication is not used, this property is ignored.</p>
User	The case-insensitive user name used to connect to your Sybase database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name.

Database-Specific Query Functions

One way to query XML data on relational databases relies on database-specific XML query features and their integration into Stylus XQuery. The advantage of this approach is it leverages database capabilities (for example, XML indexing) and allows Stylus XQuery to retrieve only the result of the query expression that is applied to the XML column.

The disadvantages of this approach are that XQuery you write for data stored on one database is not portable across different databases and database versions, and that the XML query capabilities of some databases are limited.

Note: The namespace for using database-specific XML functions in Stylus XQuery is `ddtek-sql`.

This section covers the following topics:

- [“Querying XML on DB2” on page 484](#)
- [“Querying XML on Oracle” on page 490](#)
- [“Querying XML on Microsoft SQL Server 2005” on page 497](#)

Tip: If your database supports XML Type, see [“Querying XML Type Data” on page 263](#) for alternatives to using database-specific functions to query XML data on relational databases.

Querying XML on DB2

As mentioned in [“Supported Databases” on page 445](#), IBM DB2 databases use different storage mechanisms for XML data – pureXML and XML Extender. The following table summarizes the different ways you can query XML data on DB2 depending on the type of XML storage being used.

Table 12-10. Ways to Query XML Data on DB2		
Storage Mechanism	Support for XML Type	You Can Use
pureXML	Yes	<ul style="list-style-type: none">■ Stylus XQuery. See “Querying XML Type Data”.■ DB2XML functions. See “Query Functions for DB2 XML Extender”.
XML Extender	No	<ul style="list-style-type: none">■ DB2 extract functions. See “Query Functions for DB2 pureXML”.

Query Functions for DB2 XML Extender

Stylus XQuery provides several extract functions that you can use to query XML data on DB2 when the XML Extender mechanism is in use. These functions are:

- `“ddtek-sql:DB2XML.extracttype()”`
- `“ddtek-sql:DB2XML.extracttypes()”`
- `“ddtek-sql:DB2XML.extractCLOB()”`
- `“ddtek-sql:DB2XML.extractCLOBS()”`

ddtek-sql:DB2XML.extracttype()

`extract[Integer|Smallint|Double|Real|Char|Varchar|Date|Time|Timestamp]`

This set of DB2 extract functions (`extractInteger`, `extractSmallint`, and so on) extracts the element content or attribute value from

an XML element or attribute node and returns the data as the type indicated by the function's name. For example:

```
declare function

  ddtেক-sql:DB2XML.extractDouble($inp as node(),$xp as xs:string) as
    xs:double external;
for $v1 in collection('holdingsxml')/holdingsxml
return
<shares>{
  ddtেক-sql:DB2XML.extractDouble(
    $v1/holdings/*,
    '/holdings/share[@company="Progress Software"]')
}</shares>
```

This example returns:

```
<shares>23</shares>
<shares>4000000</shares>
```

ddtek-sql:DB2XML.extracttypes()

The `ddtek-sql:DB2XML.extracttypes()` function is equivalent to `ddtek-sql:DB2XML.extracttype()`, but instead of returning a single XML fragment, it returns a table.

```
extract[Integers|Smallints|Doubles|Reals|Chars|Varchars|Dates|Times|Timestamps]
```

When using the `extracttypes()` extract function, you must declare it in the `tableFunction` element of the Stylus XQuery source configuration file, for example:

```
...
<tableFunction name="EXTRACTDOUBLES">
  <resultSet>
    <column name="RETURNEDDOUBLE" schemaType="double"/>
  </resultSet>
</tableFunction>
...
```

See [“Using SQL Table Functions” on page 337](#) for more information on this topic.

The following example shows the usage of `ddtek-sql:DB2XML.extractDoubles()`:

```
declare function ddtek-sql:DB2XML.extractDoubles($inp as node(), $xp as
  xs:string)
  as document-node(element()) external;
for $v1 in collection('holdingsxml')/holdingsxml
for $v2 in
ddtek-sql:DB2XML.extractDoubles(
  $v1/holdings/*,
  '/holdings/share')/EXTRACTDOUBLES/RETURNEDDOUBLE/data(.)
return $v2
```

This example returns:

```
3000 4000 2500 23 3000 4000 40000 4.0E6
```

ddtek-sql:DB2XML.extractCLOB()

As stated in the IBM DB2 documentation, the `extractCLOB` function:

... extracts a fragment of XML documents, with element and attribute markup and content of elements and attributes, including sub-elements. This function differs from the other extract functions, which return only the content of elements and attributes. The `extractClob(s)` functions are used to extract document fragments, whereas `extractVarchar(s)` and `extractChar(s)` are used to extract simple values.

For example:

```
declare function ddtek-sql:DB2XML.extractCLOB(
  $inp as node(), $xp as xs:string) as node() external;
for $v1 in collection('holdingsxml')/holdingsxml
return
ddtek-sql:DB2XML.extractCLOB(
  $v1/holdings/*,
  '/holdings/share[@company="Progress Software"]')
```


This example returns:

```
<share company="Progress Software" userid="Jonathan">23</share>
<share company="Progress Software" userid="Minollo">4000000</share>
```

ddtek-sql:DB2XML.extractCLOBS()

The `extractCLOBS` function is equivalent to `extractCLOB`, but instead of returning a single XML fragment, `extractCLOBS` returns a table.

In order to use `extractCLOBS`, you must declare the function in the Stylus XQuery source configuration file as a table function.

```
...
<tableFunction name="EXTRACTCLOBS">
  <resultSet>
    <column name="RETURNEDCLOB" schemaType="anyType"/>
  </resultSet>
</tableFunction>
...
```

See [“Using SQL Table Functions” on page 337](#) for more information on this topic.

The following example shows the usage of `ddtek-sql:DB2XML.extractCLOBS()`:

```
declare function ddtek-sql:DB2XML.extractCLOBS($inp as node(), $xp as
xs:string)
  as document-node(element()) external;
for $v1 in collection('holdingsxml')/holdingsxml
for $v2 in
  ddtek-sql:DB2XML.extractCLOBS(
    $v1/holdings/*,
    '/holdings/share/@company')/EXTRACTCLOBS/RETURNEDCLOB
return $v2
```

This example returns:

```
<RETURNEDCLOB>
  <share company="Amazon.com, Inc." userid="Jonathan">3000</share>
</RETURNEDCLOB>
<RETURNEDCLOB>
  <share company="eBay Inc." userid="Jonathan">4000</share>
</RETURNEDCLOB>
```

Query Functions for DB2 pureXML

Among the DB2 databases supported by Stylus XQuery, DB2 pureXML is supported on DB2 V9.1, DB2 V9.5, and DB2 V9.1 for z/OS. All of these database versions also support XML Type. However, DB2 V9.1 for z/OS supports only XPath, and not XQuery.

You can use the following built-in functions to query XML Type data on DB2 databases using pureXML:

- “ddtek-sql:db2-xmlquery()”
- “ddtek-sql:db2-xmlparse()”

NOTE: If you want to use the database’s native XQuery support, the DB2 database server must be configured with the Unicode character set.

ddtek-sql:db2-xmlquery()

The function declarations for ddtek-sql:db2-xmlquery are:

```
declare function ddtek-sql:db2-xmlquery($query as xs:string, $paramvalue as
  item()*, $paramname as xs:string) as node()* external;

declare function ddtek-sql:db2-xmlquery($query as xs:string) as
  node()* external;
```

The following example shows the use of the built-in DB2 XQuery function db2-fn:xmlcolumn(*string*) to access XML Type data. In this example, the XQuery expression is querying the holdings column of the holdingsxml table. The holdings column contains XML Type data.

```
ddtek-sql:db2-xmlquery('
  for $share in db2-fn:xmlcolumn("holdingsxml.holdings")//SHARE
  [@COMPANY="Progress Software"]
  return <progress-shares from="{ $share/@userid}" number="{data($share)}"/>
')
```

Example: ddtek-sql:db2-xmlquery() using external variables

The following example shows the use of an external variable (\$var) to access XML Type data:

```
for $v1 in collection('holdingsxml')/holdingsxml/holdings
return ddtek-sql:db2-xmlquery('
  for $v in $var//SHARE[@COMPANY="Progress Software"]
  return <progress-shares from="{ $v/@userid}" number="{data($v)}"/>',
  $v1/node(), "var")
```

A similar example using two external variables, \$var and \$companyName:

```
for $v1 in collection('holdingsxml')/holdingsxml/holdings
return ddtek-sql:db2-xmlquery('
  for $v in $var//share[@company=$CompanyName]
  return <progress-shares from="{ $v/@userid}" number="{data($v)}"/>',
  $v1/node(), "var",
  "Progress Software", "CompanyName")
```

ddtek-sql:db2-xmlparse()

The function declaration for ddtek-sql:db2-xmlparse() is:

```
declare function ddtek-sql:db2-xmlparse($doc as xs:string) as
  node() * external;
```

Here is an example:

```
let $data := '
  <holdings>
    <SHARE COMPANY="Amazon.com, Inc."
      userid="Jonathan">00003000.00</share>
    <share company="eBay Inc." userid="Jonathan">00004000.00</share>
    <share company="Int'l Business Machines C"
      userid="Jonathan">00002500.00</share>
    <share company="Progress Software"
      userid="Jonathan">00000023.00</share>
```

```

    <share company="Amazon.com, Inc."
      userid="Minollo">00003000.00</share>
    <share company="eBay Inc." userid="Minollo">00004000.00</share>
    <share company="Lucent Technologies Inc."
      userid="Minollo">00040000.00</share>
    <share company="Progress Software"
      userid="Minollo">04000000.00</share>
  </holdings>
return ddtেক-sql:db2-xmlquery('
  for $share in $p//share[@company="Progress Software"]
  return <progress-shares from="{($share/@userid)" number="{data($share) }"/> ',
  ddtেক-sql:db2-xmlparse($data), "p")

```

Querying XML on Oracle

Oracle supports XPath-like queries on XML data stored in an XML Type column through these built-in SQL extension functions:

- “ddtek-sql:existsNode()”
- “ddtek-sql:extractValue()”
- “ddtek-sql:extract()”
- “ddtek-sql:xmlSequence()”
- “ddtek-sql:ora-xmlquery()”

ddtek-sql:existsNode()

ddtek-sql:existsNode() accepts an XML value and an XPath expression as input and returns true if the XPath expression matches one or more nodes in the XML value. Refer to your Oracle documentation for details.

```

declare function ddtেক-sql:existsNode($inp as node(),$xp as xs:string) as
  xs:boolean external;
for $share in ("Amazon.com, Inc.", "eBay")
let $numberofshares :=
  count(for $x in collection('holdingsxml')/holdingsxml
    where ddtেক-sql:existsNode(
      $x/holdings/*,
      concat("/holdings/share/@company[.='", $share, "']"))
  return $x)

```

```
return <number-of-shareholders share="{ $share}" number="{ $numberofshares}" />
```

This example returns:

```
<number-of-shareholders share="Amazon.com, Inc." number="2"/>
<number-of-shareholders share="eBay" number="0"/>
```

ddtek-sql:extractValue()

ddtek-sql:extractValue() accepts an XML value and an XPath expression as input and returns the scalar value of the node selected by the XPath expression. Refer to your Oracle documentation for details.

```
declare function ddtek-sql:existsNode($inp as node(), $xp as xs:string) as
  xs:boolean external;
declare function ddtek-sql:extractValue($inp as node(), $xp as xs:string) as
  xs:untypedAtomic external;
for $share in ("Progress Software", "eBay")
for $x in collection('holdingsxml')/holdingsxml
where ddtek-sql:existsNode(
  $x/holdings/*,
  concat("/holdings/share/@company[.='", $share, "']"))
return
  <share share='{ $share}'
    holder='{ddtek-sql:extractValue(
      $x/holdings/*,
      fn:concat("/holdings/share[@company='", $share , "']/@userid"))}'
    number='{ddtek-sql:extractValue(
      $x/holdings/*,
      fn:concat("/holdings/share[@company='", $share , "']"))}' />
```

This example returns:

```
<share share="Progress Software" holder="Jonathan" number="23"/>
<share share="Progress Software" holder="Minollo" number="4000000"/>
```

ddtek-sql:extract()

ddtek-sql:extract() accepts an XML value and an XPath expression as input and returns an XML fragment that contains

the nodes selected by the XPath expression. Refer to your Oracle documentation for more details.

```
declare function ddtek-sql:extract($inp as node(), $xp as xs:string) as
  node() external;
declare function ddtek-sql:existsNode($inp as node(), $xp as xs:string) as
  xs:boolean external;
for $share in ("Amazon.com, Inc.", "Progress Software", "eBay")
for $x in collection('holdingsxml')/holdingsxml
where ddtek-sql:existsNode(
  $x/holdings/*,
  concat("/holdings/share/@company[.='", $share, "']"))
return
ddtek-sql:extract($x/holdings/*, fn:concat("/holdings/share[@company='",
  $share, "']"))
```

This example returns:

```
<share company="Amazon.com, Inc." userid="Jonathan">3000</share>
<share company="Progress Software" userid="Jonathan">23</share>
<share company="Amazon.com, Inc." userid="Minollo">3000</share>
<share company="Progress Software" userid="Minollo">4000000</share>
```

Because the result of `ddtek-sql:extract()` is a single XML fragment, limitations exist as to where this function can be used in an XQuery expression. The following usages of `ddtek-sql:extract()` are allowed:

- As a returned value from the expression, as shown in [“ddtek-sql:extract\(\)” on page 491](#)
- As input to another Oracle SQL function (`existsNode()`, `extractValue()`, `extract()`), as shown in [“Example: As Input to Another Oracle SQL Function” on page 493](#)
- Inside a `ddtek:evaluate-in-memory` extension expression, as shown in [“Example: Inside a ddtek:evaluate-in-memory Extension Expression” on page 494](#)
- As input to the Oracle `ddtek-sql:xmlSequence()` (a table function), as shown in [“ddtek-sql:xmlSequence\(\)” on page 494](#)

Here is an example of using `ddtek-sql:extract()` that is **not** allowed:

```
declare function ddtek-sql:extract($inp as node(),$xp as xs:string) as
  node() external;
declare function ddtek-sql:existsNode($inp as node(),$xp as xs:string) as
  xs:boolean external;
for $share in ("Amazon.com, Inc.,"Progress Software","eBay")
for $v1 in collection('holdingsxml')/holdingsxml
for $v2 in ddtek-sql:extract($v1/holdings/*,"/holdings/share")
where xs:string($v2/@userid) = "Minollo"
return $v2
```

This example fails and Stylus XQuery returns the following error:

```
[Stylus][XQuery]The value of the XML column "holdings" can only be used as a
return value.
```

Example: As Input to Another Oracle SQL Function

The following example shows how to use the result of `ddtek-sql:extract()` as input to `ddtek-sql:extractValue()`:

```
declare function ddtek-sql:extract($inp as node(),$xp as xs:string) as
  node() external;
declare function ddtek-sql:extractValue($inp as node(),$xp as xs:string)
  as xs:untypedAtomic external;
for $v1 in collection('holdingsxml')/holdingsxml
let $v2 := ddtek-sql:extract($v1/holdings/*,"/holdings/share")
let $v3 := ddtek-sql:extractValue($v2,
  '/share[@company="Progress Software"]/@userid')
where xs:string($v3) = "Minollo"
return $v2
```

Example: Inside a `ddtek:evaluate-in-memory` Extension Expression

This example shows how to use the result of `ddtek-sql:extract()` with the `ddtek:evaluate-in-memory` extension expression.

```
declare function ddtek-sql:extract($inp as node(), $xp as xs:string) as
  node() external;
for $v1 in collection('holdingsxml')/holdingsxml
let $v2 := ddtek-sql:extract($v1/holdings/*, '/holdings')
for $v3 in (# ddtek:evaluate-in-memory #) {$v2/share}
return
  <shares from='{ $v3/@userid}' for='{ $v3/@company}' number='{ $v3}' />
```

`ddtek-sql:xmlSequence()`

`ddtek-sql:xmlSequence()` accepts an XML value (fragment) as input and transforms it into a sequence. This is useful if you want to iterate over the result of a previously invoked `ddtek-sql:extract()`.

You must declare `ddtek-sql:xmlSequence()` in the Stylus XQuery source configuration file as a table function.

```
...
<schema name="">
  <tableFunction name="XMLSEQUENCE">
    <resultSet>
      <column name="COLUMN_VALUE" schemaType="anyType"/>
    </resultSet>
  </tableFunction>
</schema>
...
```

```
declare function ddtek-sql:extract($inp as node(), $xp as xs:string) as
  node() external;
declare function ddtek-sql:XMLSEQUENCE($inp as node()) as
  document-node(element()) external;
for $v1 in collection('holdingsxml')/holdingsxml
for $v3 in
  ddtek-sql:XMLSEQUENCE (
ddtek-sql:extract($v1/holdings/*, '/holdings/share'))/XMLSEQUENCE/COLUMN_VALUE/*
```



```
return <shareinfo>{$v3/*}</shareinfo>
```

This example returns:

```
<shareinfo>
  <share company="Amazon.com, Inc." userid="Jonathan">3000</share>
</shareinfo>
<shareinfo>
  <share company="eBay Inc." userid="Jonathan">4000</share>
</shareinfo>
...
```

ddtek-sql:ora-xmlquery()

Oracle 10g R2 and higher support native XQuery support using SQL extensions that allow combining SQL statements and XQuery expressions. Refer to your Oracle documentation for details.

Stylus XQuery supports the Oracle XMLQUERY statement using a built-in XQuery function, `ddtek-sql:ora-xmlquery`. Other Oracle XQuery related features can be supported easily through user-defined functions that wrap the SQL extensions and that can be invoked using Stylus XQuery's SQL function support. (See [“Using SQL Functions” on page 329](#).)

Stylus XQuery supports the following functional declarations of `ddtek-sql:ora-xmlquery()`:

```
declare function ddtek-sql:ora-xmlquery($query as xs:string, $context-item as
  node()?) as node()? external;

declare function ddtek-sql:ora-xmlquery($query as xs:string) as
  node()? external;
```

Using the Oracle XQuery engine, the function evaluates the XQuery expression passed by the \$query parameter. The initial context item from the XQuery expression is initialized with the value of \$context-item (Refer to your Oracle documentation for details about the meaning and usage of the initial context item). For example:

```
for $x in collection('holdingsxml')/holdingsxml/holdings
return
  ddtex-sql:ora-xmlquery(
    'for $v in //share[@company="Progress Software"]
    return <progress-shares from="{ $v/@userid}" number="{data($v)}"/>',
    $x/node()
  )
```

This example returns:

```
<progress-shares from="Jonathan" number="23"/>
<progress-shares from="Minollo" number="4000000"/>
```

Another example:

```
ddtex-sql:ora-xmlquery('ora:view("holdings")/ROW')
```

This example returns:

```
<ROW>
  <userid>Jonathan</userid>
  <stockticker>PRGS</stockticker>
  <shares>23</shares>
</ROW>
...
```

Note that the result of ddtex-sql:ora-xmlquery() is similar to the result returned from ddtex-sql:extract(). The result is an XML fragment and the same restrictions and guidelines apply.

Querying XML on Microsoft SQL Server 2005

Microsoft SQL Server 2005 introduced native XQuery support using SQL extensions that allow combining SQL statements and XQuery expressions. Refer to your Microsoft SQL Server documentation for details.

Stylus XQuery supports these Microsoft SQL Server 2005 (and higher) SQL extensions using the following proprietary (predeclared) XQuery functions:

- “ddtek-sql:sqs-query()”
- “ddtek-sql:sqs-value()”
- “ddtek-sql:sqs-exist()”
- “ddtek-sql:sqs-nodes()”

These functions correspond to the Microsoft SQL Server 2005 query(), value(), exist(), and nodes() methods.

ddtek-sql:sqs-query()

This function evaluates the Microsoft SQL Server 2005 query() method using \$context-item as the initial context item, if provided. Refer to your Microsoft SQL Server 2005 documentation for details about query().

```
declare function ddtek-sql:sqs-query($context-item as node(), $query as
  xs:string) as node()? external;
declare function ddtek-sql:sqs-query($query as xs:string) as
  node()? external;
```

For example:

```
for $x in collection('holdingsxml')/holdingsxml/holdings
return
  ddtek-sql:sqs-query(
    $x/node(),
    'for $v in //share[@company="Progress Software"]
    return <progress-shares from="{ $v/@userid}" number="{data($v)}"/>'
  )
```

This example returns:

```
<progress-shares from="Jonathan" number="23"/>
<progress-shares from="Minollo" number="4000000"/>
```

ddtek-sql:sqs-value()

This function invokes the Microsoft SQL Server 2005 `value()` method on the XML value provided by `$context-item`. Refer to your Microsoft SQL Server documentation for details about `value()`.

```
declare function ddtek-sql:sqs-value(
  $context-item as node(),
  $query as xs:string,
  $type as xs:string)
  as xs:anyAtomicType? external;
```

For example:

```
for $x in collection('holdingsxml')/holdingsxml/holdings
return
  ddtek-sql:sqs-value(
    $x/node(),
    '(for $v in //share[@company="Progress Software"] return $v)[1]', 'bigint'
  )
```

This example returns:

```
23 4000000
```

ddtek-sql:sqs-exist()

This function invokes the Microsoft SQL Server 2005 `exist()` method on the XML value provided by `$context-item`. Refer to your Microsoft SQL Server documentation for details about `exist()`.

```
declare function ddtek-sql:sqs-exist(
  $context-item as node(),
  $query as xs:string) as xs:boolean external;
```

ddtek-sql:sqs-nodes()

This is a predeclared SQL table function that returns a document node containing a sequence of `sqs-nodes` elements, each of which contains a single `col` subelement. Refer to your Microsoft SQL Server documentation for details about `node()`.

```
declare function ddtek-sql:sqs-nodes(
  $context-item as node(),
  $query as xs:string) as document-node(element()) external;
```

For example:

```
for $i in collection('holdingsxml')/holdingsxml/holdings
for $j in ddtek-sql:sqs-nodes($i/*,"//share")/sqs-nodes/col
return ddtek-sql:sqs-query($j/*,'.')
```

This example returns:

```
<share company="Amazon.com, Inc." userid="Jonathan">3000</share>
<share company="eBay Inc." userid="Jonathan">4000</share>
<share company="Int'l Business Machines C" userid="Jonathan">2500</share>
<share company="Progress Software" userid="Jonathan">23</share>
<share company="Amazon.com, Inc." userid="Minollo">3000</share>
<share company="eBay Inc." userid="Minollo">4000</share>
<share company="Lucent Technologies Inc." userid="Minollo">40000</share>
<share company="Progress Software" userid="Minollo">4000000</share>
```

Note that the result of `ddtek-sql:sqs-nodes()` can only be used as input to another `ddtek-sql:sqs-xxxx` function.

F XUF Support

This appendix describes how Stylus XQuery supports XQuery Update Facility 1.0 (XUF) expressions, functions, and operators according to this specification:

XQuery Update Facility 1.0 W3C Candidate Recommendation 1 August 2008 located at: This appendix also describes Stylus XQuery built-in functions in support of XUF.

The tables in this appendix that present XUF support information use the following terms to describe this support for XML data sources.

Term	Definition
Supported	Stylus XQuery supports the feature, function, or operator with no exceptions.
Supported with comment	Stylus XQuery supports the feature, function, or operator with the comment noted.
Not supported	Stylus XQuery does not support the feature, function, or operator and raises an error.

In this appendix, the headings and the items in the tables are numbered; these numbers correspond to the sections in the W3C Recommendation 1 August 2008 where each topic is discussed.

2 Extensions to XQuery 1.0

This section describes how Stylus XQuery supports the following extensions to XQuery 1.0.

- [“2.1 Extensions to the Processing Model” on page 502](#)
- [“2.2 Extensions to the Prolog” on page 503](#)
- [“2.3 Extensions to the Static Context” on page 503](#)
- [“2.4 New Kinds of Expressions” on page 504](#)
- [“2.5 Extensions to Existing Expressions” on page 505](#)
- [“2.6 Extensions to Built-in Function Library” on page 505](#)

2.1 Extensions to the Processing Model

[Table F-1](#) describes Stylus XQuery’s support of the extensions to the XQuery 1.0 processing model.

Table F-1. Extensions to the Processing Model

XQuery Language	Support
Extension to the Processing Model (2.1)	Supported.

2.2 Extensions to the Prolog

[Table F-2](#) describes Stylus XQuery’s support of the extensions to the XQuery 1.0 prolog.

Table F-2. Extensions to the Prolog

XQuery Language	Support
Revalidation Declaration (2.2.1)	Supported with comment. Stylus XQuery supports the <code>skip</code> setting only.
Variable Declaration (2.2.2)	Supported.
Function Declaration (2.2.3)	Supported with comment. Stylus XQuery does not support updating of external expressions.

2.3 Extensions to the Static Context

[Table F-3](#) describes Stylus XQuery’s support of the extensions to the XQuery 1.0 static context.

Table F-3. Extensions to the Static Context

XQuery Language	Support
Extensions to the Static Context (2.3)	Supported.

2.4 New Kinds of Expressions

Table F-4 describes Stylus XQuery’s support for XQuery expressions introduced by XUF.

Table F-4. New Kinds of Expressions

XQuery Language	Support
Insert (2.4.1)	Supported. Use ddtek:sql* functions to update relational database sources. See “Updating Relational Data” on page 267 for more information.
Delete (2.4.2)	Supported. Use ddtek:sql* functions to update relational database sources. See “Updating Relational Data” on page 267 for more information.
Replace (2.4.3)	
Replacing a Node (2.4.3.1)	Supported. Use ddtek:sql* functions to update relational database sources. See “Updating Relational Data” on page 267 for more information.
Replacing the Value of a Node (2.4.3.2)	Supported. Use ddtek:sql* functions to update relational database sources. See “Updating Relational Data” on page 267 for more information.
Rename (2.4.4)	Supported. Use ddtek:sql* functions to update relational database sources. See “Updating Relational Data” on page 267 for more information.
Transform (2.4.5)	Supported.
Compatibility of Updating Expressions (2.4.6)	Supported.

2.5 Extensions to Existing Expressions

Table F-5 describes Stylus XQuery’s support for extensions to existing expressions in XQuery 1.0.

Table F-5. Extensions to Built-in Function Library

XQuery Language	Support
FLWOR Expression (2.5.1)	Supported.
Typeswitch Expression (2.5.2)	Supported.
Conditional Expression (2.5.3)	Supported.
Comma Expression (2.5.4)	Supported.
Parenthesized Expression (2.5.5)	Supported.
Function Call (2.5.6)	Supported.
Other Expressions (2.5.4)	Supported.

2.6 Extensions to Built-in Function Library

Table F-6 describes Stylus XQuery’s support for extensions to the XQuery 1.0 built-in function library.

Table F-6. Extensions to Built-in Function Library

XQuery Language	Support
fn:put () (2.6.1)	Supported.

5 Conformance

Stylus XQuery fulfills the requirements of XUF Minimal Conformance (5.1).

5.2 Optional Features

Stylus XQuery supports the Update Facility Static Typing Feature (5.2.1).

G XQJ Support

This appendix describes how Stylus XQuery supports XQuery API for Java (XQJ) classes, interfaces, and methods according to the XQJ specification, which is located at:

<http://www.jcp.org/en/jsr/detail?id=225>

For additional information about the XQJ classes, interfaces, and methods, refer to the [Javadoc](#).

The tables that present XQJ support information use the following terms to describe this support:

Term	Definition
Supported	Stylus XQuery supports the method with no exceptions.
Supported with comment	Stylus XQuery supports the method with the comment noted.
Supported but ignored	Stylus XQuery does not implement the method but does not throw an exception. In this case, the method is optional in the XQJ specification.
Not supported	Stylus XQuery does not support the method and throws an exception (raises an error).

This appendix also discusses exception handling, serialization, multi-threading, accessing XML results, and mapping data types.

Java Package Name

The Java package for the XQJ interfaces is `javax.xml.xquery`.

XQConnection Interface

This interface describes the connection used to execute XQuery expressions. This interface extends [XQDataFactory](#).

[Table G-1](#) describes how Stylus XQuery supports the methods of the XQConnection interface.

Table G-1. XQConnection Method Summary

Method	Support
<code>void close()</code>	Supported.
<code>void commit()</code>	Supported. “Understanding the Transactional Behavior of Stylus XQuery Updates” on page 270.
<code>XQExpression createExpression()</code>	Supported.
<code>XQExpression createExpression(XQStaticContext staticContext)</code>	Supported.
<code>boolean getAutoCommit()</code>	Supported. See “Understanding the Transactional Behavior of Stylus XQuery Updates” on page 270 for information about auto-commit.
<code>XQMetaData getMetaData()</code>	Supported.
<code>XQStaticContext getStaticContext()</code>	Supported.
<code>String[] getSupportedMetaDataPropertyNames()</code>	Not supported.
<code>XQWarning getWarnings()</code>	Supported. Always returns null. Stylus XQuery does not generate warnings.
<code>boolean isClosed()</code>	Supported.

Table G-1. XQConnection Method Summary (*cont.*)

Method	Support
XQPreparedExpression prepareExpression(InputStream xquery)	Supported. When the query is read from a java.io.InputStream, Stylus XQuery assumes the encoding specified in the XQuery version declaration. Otherwise, Stylus XQuery parses the byte stream using UTF-8.
XQPreparedExpression prepareExpression(InputStream xquery, XQStaticContext staticContext)	Supported. When the query is read from a java.io.InputStream, Stylus XQuery assumes the encoding specified in the XQuery version declaration. Otherwise, Stylus XQuery parses the byte stream using UTF-8.
XQPreparedExpression prepareExpression(Reader xquery)	Supported.
XQPreparedExpression prepareExpression(Reader xquery, XQStaticContext staticContext)	Supported.
XQPreparedExpression prepareExpression(String xquery)	Supported.
XQPreparedExpression prepareExpression(String xquery, XQStaticContext staticContext)	Supported.
void rollback()	Supported. See “Understanding the Transactional Behavior of Stylus XQuery Updates” on page 270.
void setAutoCommit(boolean autoCommit)	Supported. See “Understanding the Transactional Behavior of Stylus XQuery Updates” on page 270 for information about auto-commit.
void setStaticContext(XQStaticContext staticContext)	Supported.

XQDataFactory

This interface represents a factory to obtain sequences, item objects, and types, which when obtained are independent of any connection.

Some of the methods in this interface create element, document, or attribute nodes. Stylus XQuery creates untyped instances of these nodes.

[Table G-2](#) describes how Stylus XQuery supports the methods of the XQDataFactory interface.

Table G-2. XQDataFactory Method Summary

Method	Support
XQItemType createAtomicType(int basetype)	Supported.
XQItemType createAtomicType(int basetype, QName typename, URI schemaURI)	Supported. Stylus XQuery does not support user-defined XML Schema types.
XQItemType createAttributeType(QName nodename, int basetype)	Supported.
XQItemType createAttributeType(QName nodename, int basetype, QName typename, URI schemaURI)	Supported. Stylus XQuery does not support user-defined XML Schema types. ^a
XQItemType createCommentType()	Supported.
XQItemType createDocumentElementType (XQItemType elementType)	Supported.
XQItemType createDocumentSchemaElementType (XQItemType elementType)	Stylus XQuery does not support user-defined XML Schema types.
XQItemType createDocumentType()	Supported.
XQItemType createElementType(QName nodename, int basetype)	Supported.

Table G-2. XQDataFactory Method Summary *(cont.)*

Method	Support
XQItemType createElementType(QName nodename, int basetype, QName typename, URI schemaURI, boolean allowNull)	Supported. schemaURI and allowNull are ignored. If typename is not null, then it must match the value specified for basetype. ¹
XQItem createItem(XQItem item)	Supported.
XQItem createItemFromAtomicValue(String value, XQItemType type)	Supported.
XQItem createItemFromBoolean(boolean value, XQItemType type)	Supported.
XQItem createItemFromByte(byte value, XQItemType type)	Supported.
XQItem createItemFromDocument(InputStream value, String baseURI, XQItemType type)	Supported.
XQItem createItemFromDocument(Reader value, String baseURI, XQItemType type)	Supported.
XQItem createItemFromDocument(String value, String baseURI, XQItemType type)	Supported.
XQItem createItemFromDocument(Source value, XQItemType type)	Supported.
XQItem createItemFromDocument(XMLStreamReader value, XQItemType type)	Supported.
XQItem createItemFromDouble(double value, XQItemType type)	Supported.
XQItem createItemFromFloat(float value, XQItemType type)	Supported.
XQItem createItemFromInt(int value, XQItemType type)	Supported.
XQItem createItemFromLong(long value, XQItemType type)	Supported.
XQItem createItemFromNode(Node value, XQItemType type)	Supported.
XQItem createItemFromObject(Object value, XQItemType type)	Supported.

Table G-2. XQDataFactory Method Summary *(cont.)*

Method	Support
XQItem createItemFromShort(short value, XQItemType type)	Supported.
XQItem createItemFromString(String value, XQItemType type)	Supported.
XQItemType createItemType()	Supported.
XQItemType createNodeType()	Supported.
XQItemType createProcessingInstructionType(String piTarget)	Supported.
XQItemType createSchemaAttributeType(QName nodename, int basetype, URI schemaURI)	Stylus XQuery does not support user-defined XML Schema types.
XQItemType createSchemaElementType(QName nodename, int basetype, URI schemaURI)	Stylus XQuery does not support user-defined XML Schema types.
XQSequence createSequence(Iterator i)	Supported.
XQSequence createSequence(XQSequence s)	Supported.
XQSequenceType createSequenceType(XQItemType item, int occurrence)	Supported.
XQItemType createTextType()	Supported.

a. An example where typename does not equal basetype:

```

createAttributeType(
    new QName("http://www.foo.com", "bar"),
    XQItemType.XQBASETYPE_INTEGER,
    new QName("http://www.w3.org/2001/XMLSchema", "decimal"),null);
  
```

XQDataSource Interface

This interface can be used as a factory for creating XQuery connection objects.

A DataSource object typically has a set of properties that identify and describe the data source that it represents such as location of the database, the name of the database, and so on.

The class name of the Stylus XQuery XQDataSource implementation is:

`com.ddtek.xquery.xqj.DDXQDataSource`

In addition, Stylus XQuery uses the following class to specify additional properties when configuring connections for multiple databases:

`com.ddtek.xquery.xqj.DDXQJDBCConnection`

See [“Configuring Connections Explicitly” on page 123](#) for information about configuring connections using the DDXQDataSource and DDXQJDBCConnection interfaces, and the properties supported by DDXQDataSource and DDXQJDBCConnection.

[Table G-3](#) describes how Stylus XQuery supports the methods of the XQDataSource interface.

Table G-3. XQDataSource Method Summary

Method	Support
XQConnection getConnection()	Supported.
XQConnection getConnection(Connection con)	Not supported.
XQConnection getConnection(String username, String passwd)	Supported. The user name and password value pair is used for all underlying JDBC connections.
int getLoginTimeout()	Supported. Always returns 0, which means there is no timeout.
PrintWriter getLogWriter()	Supported.
String getProperty(String name)	Supported. See Table 6-1 on page 128 for a description of all supported properties.
String[] getSupportedPropertyNames()	Supported. See Table 6-1 on page 128 for a description of all supported properties.
void setLoginTimeout(int seconds)	Supported but ignored.

Table G-3. XQDataSource Method Summary *(cont.)*

Method	Support
void setLogWriter(PrintWriter out)	Supported.
void setProperties(Properties props)	Supported. See Table 6-1 on page 128 for a description of all supported properties.
void setProperty(String name, String value)	Supported. See Table 6-1 on page 128 for a description of all supported properties.

XQDynamicContext Interface

This interface provides get and bind methods for the components of the dynamic context of an XQuery expression. This interface is implemented through XQExpression and XQPreparedExpression.

[Table G-4](#) describes how Stylus XQuery supports the methods of the XQDynamicContext interface.

Table G-4. XQDynamicContext Method Summary

Method	Support
void bindAtomicValue(QName varName, String value, XQItemType type)	Supported.
void bindBoolean(QName varName, boolean value, XQItemType type)	Supported.
void bindByte(QName varname, byte value, XQItemType type)	Supported.
void bindDocument(QName varName, InputStream value, String baseURI, XQItemType type)	Supported.
void bindDocument(QName varName, Reader value, String baseURI, XQItemType type)	Supported.

Table G-4. XQDynamicContext Method Summary *(cont.)*

Method	Support
void bindDocument(namespaceQName varName, Source value, XQItemType type)	Supported.
void bindDocument(QName varName, String value, String baseURI, XQItemType type)	Supported.
void bindDocument(QName varName, XMLStreamReader value, XQItemType type)	Supported.
void bindDouble(QName varName, double value, XQItemType type)	Supported.
void bindFloat(QName varName, float value, XQItemType type)	Supported.
void bindInt(QName varName, int value, XQItemType type)	Supported.
void bindItem(QName varName, XQItem value)	Supported.
void bindLong(QName varName, long value, XQItemType type)	Supported.
void bindNode(QName varName, Node value, XQItemType type)	Supported.
void bindObject(QName varName, Object value, XQItemType type)	Supported.
void bindSequence(QName varName, XQSequence value)	Supported.
void bindShort(QName varName, short value, XQItemType type)	Supported.
void bindString(QName varName, String value, XQItemType type)	Supported.
TimeZone getImplicitTimeZone()	Not supported.
void setImplicitTimeZone(TimeZone implicitTimeZone)	Not supported.

XQExpression Interface

This interface describes how an XQuery expression will be executed. This object can be created from the XQConnection object and the execution can be performed using the executeQuery function or executeCommand method, passing in the XQuery expression. This interface extends [XQDynamicContext](#).

[Table G-5](#) describes how Stylus XQuery supports the methods of the XQExpression interface.

Table G-5. XQExpression Method Summary

Method	Support
void cancel()	Supported but ignored.
void close()	Supported.
void executeCommand(Reader cmd)	Supported. However, Stylus XQuery does not have any proprietary commands. Always throws an exception.
void executeCommand(String cmd)	Supported. However, Stylus XQuery does not have any proprietary commands. Always throws an exception.
XQResultSequence executeQuery (InputStream xquery)	Supported. If the query is read from a java.io.InputStream, Stylus XQuery assumes the encoding specified in the XQuery version declaration. Otherwise, Stylus XQuery parses the byte stream using UTF-8.
XQResultSequence executeQuery(Reader query)	Supported.
XQResultSequence executeQuery(String query)	Supported.
XQStaticContext getStaticContext()	Supported.
boolean isClosed()	Supported.

XQItem Interface

This interface represents an item in the XDM (XQuery 1.0 and XPath 2.0 Data Model). This interface extends [XQItemAccessor](#).

[Table G-6](#) describes how Stylus XQuery supports the methods of the XQItem interface.

Table G-6. XQItem Method Summary

Method	Support
void close()	Supported.
boolean isClosed()	Supported.

XQItemAccessor Interface

This interface represents a common interface for accessing the values of an XQuery item.

[Table G-7](#) describes how Stylus XQuery supports the methods of the XQItemAccessor interface.

Table G-7. XQItemAccessor Method Summary

Method	Support
String getAtomicValue()	Supported.
boolean getBoolean()	Supported.
byte getByte()	Supported.
double getDouble()	Supported.
float getFloat()	Supported.

Table G-7. XQItemAccessor Method Summary *(cont.)*

Method	Support
int getInt()	Supported.
XMLStreamReader getItemAsStream()	Supported.
String getItemAsString(Properties props)	Supported.
XQItemType getItemType()	Supported.
long getLong()	Supported.
Node getNode()	Supported.
URI getNodeUri()	Not supported. Always throws UnsupportedOperationException.
Object getObject()	Supported as explained in Table G-18 “Mapping XQuery Data Model Instances to Java Objects” on page 536.
short getShort()	Supported.
boolean instanceOf(XQItemType type)	Supported. When an unknown type is specified, false is returned.
void writeItem(OutputStream os, Properties props)	Supported.
void writeItem(Writer ow, Properties props)	Supported.
void writeItemToResult(Result result)	Supported.
writeItemToSAX(ContentHandler saxhdlr)	Supported.

XQItemType Interface

The XQItemType interface represents an item type as described in [XQuery 1.0: An XML Query Language](#). This interface extends [XQSequenceType](#).

[Table G-8](#) describes how Stylus XQuery supports the methods of the XQItemType interface.

Table G-8. XQItemType Method Summary

Method	Support
int getBaseType()	Supported. See “Restrictions” on page 520 .
int getItemKind()	Supported.
int getItemOccurence()	Supported.
QName getNodeName()	Supported.
String getPIName()	Supported.
URI getSchemaURI()	Supported. Always returns null.
String getString()	Supported.
QName getTypeName()	Supported. Stylus XQuery returns a QName that matches the schema built-in type. For example: <pre>XQItemType type = datafactory.createAtomicType (XQItemType.XQBASETYPE_STRING); QName typeName= type.getTypeName();</pre> The typeName is now: <pre>new QName("http://www.w3.org/2001/XMLSchema", "string")</pre> See “Restrictions” on page 520 .
boolean isAnonymousType()	Supported. Always returns false.
boolean isElementNillable()	Supported. Always returns false.
String toString()	Supported.

See [Table G-16 “XQuery Types Supported for XQJ get Methods” on page 533](#) for a list of XQuery types supported for the get methods of the XQItemType interface.

Restrictions

Because Stylus XQuery is not schema aware, a few restrictions apply to the XQItemType instances that Stylus XQuery can accept. These restrictions are:

- The value returned by `getBaseType()` must not be:
 - `XQBASETYPE_ENTITIES`
 - `XQBASETYPE_IDREFS`
 - `XQBASETYPE_NMTOKENS`
- Depending on the item kind, `getBaseType()` must return specific base types, as follows:
 - If `XQITEMKIND_ATTRIBUTE`, `getBaseType()` must return either `XQBASETYPE_ANYSIMPLETYPE` or `XQBASETYPE_UNTYPEDATOMIC`.
 - If `XQITEMKIND_ELEMENT` or `XQITEMKIND_DOCUMENT_ELEMENT`, `getBaseType()` must return either `XQBASETYPE_ANYTYPE` or `XQBASETYPE_UNTYPED`.
- The value returned by `getTypeName()` must be one of the following:
 - `null`.
 - A `QNAME` representing one of the built-in XML Schema types.
 - A value consistent with `getBaseType()`. For example, if not `null`, the specified XML Schema type must match exactly `getBaseType()`.

XQMetaData Interface

This interface provides information about the XQJ implementation, such as product name and version, supported features, user information, and more.

[Table G-9](#) describes how Stylus XQuery supports the methods of the XQMetaData interface.

Table G-9. XQMetaData Method Summary

Method	Support
int getMaxExpressionLength()	Supported. Always returns 0. There is no limit on length.
int getMaxUserNameLength()	Supported. Always returns 0. The limit is unknown.
int getProductMajorVersion()	Supported.
int getProductMinorVersion()	Supported.
String getProductName()	Supported. Returns "Stylus XQuery".
String getProductVersion()	Supported. Returns version information, including build number, and date/time.
set getSupportedXQueryEncodings()	Supported. The result is JVM dependent, returns all supported character sets of the JVM on which the application is running.
String getUsername()	Supported.
int getXQJMajorVersion()	Supported. Always returns 1.
int getXQJMinorVersion()	Supported. Always returns 0.
String getXQJVersion()	Supported. Always returns "1.0".
boolean isFullAxisFeatureSupported()	Supported. Always returns true.
boolean isModuleFeatureSupported()	Supported. Always returns true.
boolean isReadOnly()	Supported. Always returns false.
boolean isSchemaImportFeatureSupported()	Supported. Always returns false.

Table G-9. XQMetaData Method Summary *(cont.)*

Method	Support
boolean isSchemaValidationFeatureSupported()	Supported. Always returns false.
boolean isSerializationFeatureSupported()	Supported. Always returns true.
boolean isStaticTypingExtensionsSupported()	Supported. Always returns true.
boolean isStaticTypingFeatureSupported()	Supported. Always returns true.
boolean isTransactionSupported()	Supported. Always returns true.
boolean isUserDefinedXMLSchemaTypeSupported()	Supported. Always returns false.
boolean isXQueryEncodingDeclSupported()	Supported. Always returns true.
boolean isXQueryEncodingSupported (String encoding)	Supported.
boolean isXQueryXSupported()	Supported. Always returns false.
boolean wasCreatedFromJDBCCConnection()	Supported. Always returns false.

XQPreparedExpression Interface

This interface describes an expression that can be prepared for multiple subsequent executions. A prepared expression can be created from the connection. This interface extends [XQDynamicContext](#).

[Table G-10](#) describes how Stylus XQuery supports the methods of the XQPreparedExpression interface.

Table G-10. XQPreparedExpression Method Summary

Method	Support
void cancel()	Supported but ignored.
void close()	Supported.

Table G-10. XQPreparedExpression Method Summary

Method	Support
XQResultSequence executeQuery()	Supported.
QName[] getAllExternalVariables()	Supported.
QName[] getAllUnboundExternalVariables()	Supported.
XQStaticContext getStaticContext()	Supported.
XQSequenceType getStaticResultType()	Supported.
XQSequenceType getStaticVariableType(QName name)	Supported.
boolean isClosed()	Supported.

XQResultItem Interface

The XQSequenceType interface is used to represent the type information of a sequence as described in the XDM (XQuery 1.0 and XPath 2.0 Data Model). This interface extends [XQItem](#) and [XQItemAccessor](#).

[Table G-11](#) describes how Stylus XQuery supports the methods of the XQResultItem interface.

Table G-11. XQResultItem Method Summary

Method	Support
XQConnection getConnection()	Supported.

XQResultSequence Interface

This interface represents a sequence of items obtained as a result of evaluating XQuery expressions. This interface extends [XQSequence](#) and [XQItemAccessor](#).

[Table G-12](#) describes how Stylus XQuery supports the methods of the XQResultSequence interface.

Table G-12. XQResultSequence Method Summary

Method	Support
XQConnection getConnection()	Supported.

XQSequence Interface

This interface represents a sequence of items as defined in the XPath/XQuery data model. The sequence may be materialized or non-materialized. This interface extends [XQItemAccessor](#).

[Table G-13](#) describes how Stylus XQuery supports the methods of the XQSequence interface.

Table G-13. XQSequence Method Summary

Method	Support
boolean absolute(int itempos)	Supported.
void afterLast()	Supported.
void beforeFirst()	Supported.
void close()	Supported.
int count()	Supported.

Table G-13. XQSequence Method Summary (cont.)

Method	Support
boolean first()	Supported.
XQItem getItem()	Supported.
int getPosition()	Supported.
XMLStreamReader getSequenceAsStream()	Supported.
String getSequenceAsString(Properties props)	Supported as explained in Appendix D “Serialization Support” on page 439 .
boolean isAfterLast()	Supported.
boolean isBeforeFirst()	Supported.
boolean isClosed()	Supported.
boolean isFirst()	Supported.
boolean isLast()	Supported.
boolean isOnItem()	Supported.
boolean isScrollable()	Supported.
boolean last()	Supported.
boolean next()	Supported.
boolean previous()	Supported.
boolean relative(int itempos)	Supported.
void writeSequence(OutputStream os, Properties props)	Supported as explained in Appendix D “Serialization Support” on page 439 .
void writeSequence(Writer ow, Properties props)	Supported as explained in Appendix D “Serialization Support” on page 439 .
void writeSequenceToResult(Result result)	Supported.
void writeSequenceToSAX(ContentHandler saxhdlr)	Supported.

XQSequenceType Interface

This interface represents sequence type as described in [XQuery 1.0: An XML Query Language](#).

[Table G-14](#) describes how Stylus XQuery supports the methods of the XQSequenceType interface.

Table G-14. XQSequenceType Method Summary

Method	Support
int getItemOccurence()	Supported.
XQItemType getItemType()	Supported.
String toString()	Supported. The return value is formatted using the string representation of the underlying XQItemType, optionally appended with "?", "+", or "*", depending on the item occurrence.

XQStaticContext Interface

This interface represents default values for XQuery Static Context components. Additionally, it includes the static XQJ properties for an XQExpression or XQPreparedExpression object.

[Table G-15](#) describes how Stylus XQuery supports the methods of the XQStaticContext interface.

Table G-15. XQStaticContext Method Summary

Method	Support
void declareNamespace(String prefix, String uri)	Supported.
String getBaseURI()	Supported. Returns the base URI as specified in the BaseUri property of DDXQDataSource or the directory of your current JVM if no base URI is specified.
int getBindingMode()	Supported.
XQItemType getContextItemStaticType()	Supported.
int getCopyNamespacesModelInherit()	Supported. Always returns the same constant value, XQConstants.COPY_NAMESPACES_MODE_INHERIT.
int getCopyNamespacesModePreserve()	Supported. Always returns the same constant value, XQConstants.COPY_NAMESPACES_MODE_PRESERVE.
String getDefaultCollation()	Supported. Returns the collation as specified in the Collation property of DDXQDataSource or the default JVM collation if no collation is specified.
String getDefaultElementTypeNamespace()	Supported. Returns an empty string.
String getDefaultFunctionNamespace()	Supported. Returns http://www.w3.org/2005/xpath-functions
int getDefaultOrderForEmptySequences()	Supported. Returns the default order for empty sequences. See the "Default order for empty sequences" description in Table A-1 "XQuery Expression Context" on page 342 .
int getHoldability()	Supported. Always returns the same constant value. See the corresponding set method for the value returned.
String[] getNamespacePrefixes()	Supported. Returns xsi, xdt, xml, ddtex-sql, fn, ddtex, local, ddtex-sql-jdbc, and xs.
String getNamespaceURI(String prefix)	Supported.

Table G-15. XQStaticContext Method Summary *(cont.)*

Method	Support
int getOrderingMode	Supported. Always returns the same constant value, XQConstants.ORDERING_MODE_ORDERED.
int getQueryLanguageTypeAndVersion()	Supported. Always returns the same constant value, XQConstants.LANGTYPE_XQUERY
int getQueryTimeout()	Supported. Always returns 0, which means there is no timeout.
int getScrollability()	Supported. Always returns the same constant value, XQConstants.SCROLLTYPE_FORWARD_ONLY.
void setBaseURI(String baseUri)	Supported.
void setBindingMode()	Supported. See “Support of Deferred Binding” on page 532 .
void setBoundarySpacePolicy(int policy)	Supported.
void setConstructionMode(int mode)	Supported. mode must be set to XQConstants.CONSTRUCTION_MODE_PRESERVE.
void setContextItemStaticType(XQItemType contextItemType)	Supported.
void setCopyNamespacesModelInherit(int mode)	Supported. mode must be set to XQConstants.COPY_NAMESPACES_MODE_INHERIT.
void setCopyNamespacesModePreserve(int mode)	Supported. Must be set to XQConstants.COPY_NAMESPACES_MODE_PRESERVE.
void setDefaultCollation(String uri)	Supported.
void setDefaultElementTypeNamespace(String uri)	Supported.
void setDefaultFunctionNamespace(String uri)	Supported.
void setDefaultOrderForEmptySequences(int order)	Supported.
void setHoldability(int holdability)	Supported. holdability must be CURSORS_OVER_COMMIT.
void setOrderingMode(int mode)	Supported.

Table G-15. XQStaticContext Method Summary (cont.)

Method	Support
void setQueryLanguageAndVersion(int langType)	Supported. langType must be set to XQConstants.LANGTYPE_XQUERY.
void setQueryTimeout(int seconds)	Supported.
void setScrollability(int scrollability)	Supported. scrollability must be set to XQConstants.SCROLLTYPE_FORWARD_ONLY

Exception Handling

Stylus XQuery uses two XQJ classes to report errors, XQException and XQQueryException. Stylus XQuery reports errors detected in XQJ using XQException.

Errors reported through XQException contain the following information:

- Vendor code
- Message
- Cause

For example:

```
XQException: [Stylus][XQuery]Exception returned by
Adaptor (Streaming Adaptor) : Unexpected close tag
</ducks>; expected </papere>.
```

Errors detected in any of the other Stylus XQuery components can be reported using either class depending on the type of error. If the error is detected during query processing, Stylus XQuery reports the error using XQQueryException

because additional information is contained in the message. This additional information might be a line number that indicates the location of the error in the query and a module URI that indicates the module where the error occurred.

For example:

```
XQueryException:  
{http://www.w3.org/2005/xqt-errors}XPST0003  
file:///myXQuery.xq; line: 2  
[Stylus][XQuery][err:XPST0003]Error at line 2, column  
44. Expected "]", but encountered "<end of expression>".
```

Multi-Threading Support

Stylus XQuery is thread safe.

Accessing XML Results

You can access XML results of a query as:

- DOM
- SAX
- StAX
- Text

For examples showing how to access XML results, see [“Returning Results with Java XML APIs” on page 69](#).

DOM

The DOM nodes returned from XML results are DOM Level 2 Core API compliant.

SAX

Stylus XQuery serializes a result sequence or a single item as SAX events in the following way:

- 1 The instance of the data model is normalized as described in: <http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20050404/#serdm>
- 2 The normalization process results in one of the following:
 - A well-formed XML document as described in: <http://www.w3c.org/TR/2000/REC-xml-20001006#sec-well-formed>
 - A well-formed XML external parsed entity as described in: <http://www.w3c.org/TR/2000/REC-xml-20001006#wf-entities>
- 3 Events are generated as defined by the SAX2 specification.

Stylus XQuery reports comment nodes to the specified ContentHandler if that object also implements the LexicalHandler interface.

StAX

Stylus XQuery serializes a result sequence or a single item as StAX events in the following way:

- 1 The instance of the data model is normalized as described in: <http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20050404/#serdm>

- 2 The normalization process results in one of the following:
 - A well-formed XML document as described in:
<http://www.w3c.org/TR/2000/REC-xml-20001006#sec-well-formed>
 - A well-formed XML external parsed entity as described in:
<http://www.w3c.org/TR/2000/REC-xml-20001006#wf-entities>
- 3 Events are generated as defined by the StAX 1.0 specification.

Text

Stylus XQuery supports serializing query results (serialization refers to converting query results to text). The `writeSequence` and `getSequenceAsString` methods of the `XQSequence` interface allow you to serialize the results to the following Java objects: `java.lang.String`, `java.io.Writer`, and `java.io.OutputStream`. This serialization conforms to the process as described in the XSLT 2.0 and XQuery 1.0 Serialization specification, located at:
<http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20050404>

See [Appendix D “Serialization Support” on page 439](#) for more information.

Support of Deferred Binding

When configured to use deferred binding, Stylus XQuery does not use the value bound to a variable until the value is needed, and it enables Stylus XQuery to use its XML streaming feature. For such reasons, deferred binding can result in performance and scalability benefits. See [“Querying Large XML Documents” on page 177](#) for information about the XML streaming feature.

To set deferred binding, use the `setBindingMode()` method of the `XQStaticContext` interface. For example:

```
XQStaticContext cntxt = conn.getStaticContext();
cntxt.setBindingMode(XQConstants.BINDING_MODE_DEFERRED);
conn.setStaticContext(cntxt);
```

XQuery Types Supported by XQJ get Methods

For a value of an XQuery result to be used by a Java application, the value must be mapped to a Java data type using the XQJ get methods. [Table G-16](#) lists the XQuery types that are supported for XQJ get methods.

Table G-16. XQuery Types Supported for XQJ get Methods

get Method	Supported XQuery Type
<code>getAtomicValue</code>	<code>xs:anyAtomicType</code> (or any of its built-in subtypes derived by restriction)
<code>getBoolean</code>	<code>xs:boolean</code>
<code>getByte</code>	<code>xs:decimal</code> (or any of its built-in subtypes derived by restriction)
<code>getDouble</code>	<code>xs:double</code>
<code>getFloat</code>	<code>xs:float</code>
<code>getInt</code>	<code>xs:decimal</code> (or any of its built-in subtypes derived by restriction)
<code>getLong</code>	<code>xs:decimal</code> (or any of its built-in subtypes derived by restriction)
<code>getNode</code>	<code>node()</code>
<code>getObject</code>	<code>xs:anyAtomicType</code> or <code>node()</code>
<code>getShort</code>	<code>xs:decimal</code> (or any of its built-in subtypes derived by restriction)

Retrieving and Binding XQuery Data Model Instances

For each Java data type that is bound to an XQuery external variable, there is an XQDynamicContext method to bind values of that type to the XQuery expression. These methods and how they map to resulting XQuery data model instances are shown in [Table G-17](#).

Table G-17. XQJ bind Methods and Resulting XQuery Data Model Instances

bind Method	Resulting XQuery Data Model Instance
bindAtomicValue	xs:anyAtomicType (type of the specified atomic type)
bindBoolean	xs:boolean
bindByte	xs:byte
bindDocument	document-node()
bindDouble	xs:double
bindFloat	xs:float
bindInt	xs:int
bindItem	Type of the specified item
bindLong	xs:long
bindNode	element(), document-node(), attribute(), comment(), processing-instruction(), or text() NOTE: The actual node type depends on the specified org.w3c.dom.Node type.
bindObject	xs:anyAtomicType or a subtype of node()
bindSequence	For each item in the sequence, the type of the item
bindShort	xs:short
bindString	xs:string

The created data model instance is checked for compatibility with the static type of the external variable using the SequenceType matching rules. Stylus XQuery supports the following type declarations for external variables:

```
SequenceType      ::= (ItemType OccurrenceIndicator?)
                    | ("empty" "(" ")")
OccurrenceIndicator ::= "?" | "*" | "+"
ItemType          ::= AtomicType | KindTest
                    | ("item" "(" ")")
AtomicType        ::= QName
KindTest          ::= DocumentTest
                    | ElementTest
                    | AttributeTest
                    | PITest
                    | CommentTest
                    | TextTest
                    | AnyKindTest
PITest            ::= "processing-instruction" "(" ")"
CommentTest       ::= "comment" "(" ")"
TextTest          ::= "text" "(" ")"
AnyKindTest       ::= "node" "(" ")"
DocumentTest      ::= "document-node" "(" ")"
ElementTest       ::= "element" "(" ")"
AttributeTest     ::= "attribute" "(" ")"
```

[Table G-18](#) describes how XQuery data model instances map to Java objects when the following XQJ methods are being used:

- getObject() of XQItemAccessor interface. This method returns a Java object corresponding to the corresponding XQuery data model instance.
- bindObject() of XQDynamicContext interface. This method binds a XQuery data model instance to a Java object.
- createItemFromObject() of XQDynamicContext interface. This method creates an XQuery data model instance from a Java object.

Table G-18. Mapping XQuery Data Model Instances to Java Objects

XQuery Data Model Instance	Java Object
document node	org.w3c.dom.Document
element node	org.w3c.dom.Element
attribute node	org.w3c.dom.Attr
comment node	org.w3c.dom.Comment
processing-instruction node	org.w3c.dom.ProcessingInstruction
text node	org.w3c.dom.Text
xs:untypedAtomic	java.lang.String
xs:string	java.lang.String
xs:boolean	java.lang.Boolean
xs:decimal	java.math.BigDecimal
xs:float	java.lang.Float
xs:double	java.lang.Double
xs:dateTime	javax.xml.datatype.XMLGregorianCalendar ^a
xs:date	javax.xml.datatype.XMLGregorianCalendar ¹
xs:time	javax.xml.datatype.XMLGregorianCalendar ¹
xs:integer	java.math.BigInteger
xs:long	java.lang.Long
xs:int	java.lang.Integer
xs:short	java.lang.Short
xs:byte	java.lang.Byte
xs:nonPositiveInteger	java.math.BigInteger
xs:negativeInteger	java.math.BigInteger
xs:nonNegativeInteger	java.math.BigInteger
xs:unsignedLong	java.math.BigInteger
xs:unsignedInt	java.lang.Long
xs:unsignedShort	java.lang.Integer
xs:unsignedByte	java.lang.Short
xs:positiveInteger	java.math.BigInteger

Table G-18. Mapping XQuery Data Model Instances to Java Objects (cont.)

XQuery Data Model Instance	Java Object
xs:base64binary	byte[]
xs:hexBinary	byte[]
xs:duration	javax.xml.datatype.Duration ¹
xs:yearMonthDuration	javax.xml.datatype.Duration ¹
xs:dayTimeDuration	javax.xml.datatype.Duration ¹
xs:QName	java.xml.namespace.QName
xs:anyURI	java.net.URI
xs:gDay	javax.xml.datatype.XMLGregorianCalendar ¹
xs:gMonth	javax.xml.datatype.XMLGregorianCalendar ¹
xs:gMonthDay	javax.xml.datatype.XMLGregorianCalendar ¹
xs:gYear	javax.xml.datatype.XMLGregorianCalendar ¹
xs:gYearMonth	javax.xml.datatype.XMLGregorianCalendar ¹
xs:normalizedString	java.lang.String
xs:token	java.lang.String
xs:language	java.lang.String
xs:NMTOKEN	java.lang.String
xs:Name	java.lang.String
xs:NCName	java.lang.String
xs:ID	java.lang.String
xs:IDREF	java.lang.String
xs:ENTITY	java.lang.String

a. When using J2SE 1.4, this XQuery data model instance cannot be retrieved through getObject() of XQItemAccessor, bound through bindObject() of XQDynamicContext, or created through createItemFromObject() of XQDynamicContext. However, you can manipulate this data model instance using getAtomicValue() of XQItemAccessor, bindAtomicValue() of XQDynamicContext, or createItemFromAtomicValue() of XQDataFactory.

H Examples

This appendix explains the example Java applications that are shipped with Stylus XQuery and provides instructions for setting up and running them.

Required Software

This section describes the requirements that must be met before setting up and running the Stylus XQuery examples.

Database

Stylus XQuery examples support specific versions of IBM DB2, Informix, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, and Sybase. See [“Relational Data Sources” on page 118](#) for information about supported database versions.

Stylus XQuery®

You must install Stylus XQuery before using the Stylus XQuery examples. Stylus XQuery requires J2SE 1.4.x or later.

Configuring Your Environment to Run the Examples

- 1 Configure a database instance to connect to and privileges to create and drop tables. You must have a user ID and password that has permission to create tables, drop tables, and insert rows of data. The data loader provided with the examples will create and drop tables used by the examples.

IMPORTANT: The examples data loader drops and creates the following database tables: "users", "companies", "historical", "holdings", and "statistical".

- 2 Modify the `setenv.bat` (Windows) or `setenv.sh` (UNIX/Linux) file. In addition, you must set the `JDBCURL` environment setting in the file to the connection URL of the test database to which you want to connect. See the comments in the `setenv` file for examples.

Optionally, you may need to set:

- The `PATH2LIBS` environment setting to the Stylus XQuery lib subdirectory:

ddxq_install_directory/lib

If you are executing the examples and `dataloader` from their respective directories, you do not need to modify this value.

- The `PATH` variable so that the Java executable is available from the command line. If you must recompile the examples, `PATH` should also contain the path that specifies the location of the Java compiler (`javac`) executable.

3 Populate the test database. To load the data, execute:

```
ddxq_install_directory/examples/DataLoader/  
load_example_data.bat (Windows)
```

or

```
ddxq_install_directory/examples/DataLoader/  
load_example_data.sh (UNIX/Linux)
```

Creation of the database table and data upload may take as few as 30 seconds; however, depending on the speed of your network, database server, and workstation, the process may take several minutes.

NOTES:

- The data loader will create the following database tables: "users", "companies", "historical", "holdings", and "statistical".
- Using the data loader with a Java VM that was started with the -server option can crash the Java VM. This is caused by a Java bug that can occur with any supported JDK version and on different platforms. For information about workarounds for this issue, contact Technical Support.

About the Examples

Each example is located in a separate directory beneath the examples directory:

```
ddxq_install_directory/examples/name_of_example
```

For example:

```
ddxq5_0/examples/xqjexecute
```

Each example directory contains the following files:

- `run.bat` or `run.sh` to execute the example program. This file contains a commented line, that when uncommented, recompiles the example.
- Zero or more `.xq` files that contain the XQuery expressions that will be executed by the example program.
- Zero or more `.xml` files that contain XML files that will be queried by the XQuery expressions.
- One or more source and class files that contain the source and compiled source code for the example.

The following examples are provided with the product:

- [Connect](#) demonstrates some of Stylus XQuery's advanced connection options.
- [CustomDocumentURIResolver](#) shows how to implement a custom URI resolver.
- [JNDIDataSource](#) shows how to persist and load a `DDXQDataSource` using a JNDI provider.
- [ExternalFunctions](#) shows different types of external functions supported by Stylus XQuery and how they can be used.
- [ExternalVariables](#) shows binding Java variables to external XQuery variables.
- [RDBMSUpdate](#) shows how to insert, update, and delete data stored in a relational database.
- [ResultRetrieval](#) shows how you can retrieve the results of an XQuery expression as SAX, StAX, or DOM.
- [UpdateFacility](#) shows how to update data in XML documents.
- [XMLQuery](#) shows how to query data in XML documents.

- [XQJExecute](#) is a simple XQJ example that shows how to execute an XQuery expression from a file or string and retrieve the results as a string.

For information about how to use each of these examples, see the following sections.

Connect

Before using this example, you must edit two source configuration files located in the Connect directory:

- config_basic.xml
- config_advanced.xml

You must specify the correct values for the following elements in these files: url, user, and password. In addition, in the config_advanced.xml file, you must enter the correct values for the name attributes for the catalog and schema elements. See [“Using a Source Configuration File” on page 533](#) for more information about source configuration files.

From the Connect directory, enter the following command line to execute the example:

Windows:

```
run.bat
```

UNIX:

```
run.sh
```

A prompt appears asking you to enter one of the following numbers that corresponds to the connection method you want to use:

- 1 - Connect using config_basic.xml source configuration file.
- 2 - Connect using config_advanced.xml source configuration

- file, this example shows the use of a table alias.
- 3 - Connect using config_advanced.xml source configuration file, this example shows the use of a target namespace for a database table.
 - 4 - Connect using config_advanced.xml source configuration file, this example shows how to eliminate certain table columns from the SQL/XML view of the table.
 - 5 - Connect using config_advanced.xml source configuration file, this example shows the use of a base URI.
 - 6 - Connects using config_advanced.xml source configuration file, this example illustrates the use of Stylus Spy for XQJ.

CustomDocumentURIResolver

From the CustomURIResolver directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where *xquery_file* is the XQuery file you want to execute. If you do not specify a query file name, the XQuery file in the CustomURIResolver directory is executed. Optionally, you can write your own query, save it to a file, and execute that XQuery file by entering the file name as the argument to run.bat or run.sh.

This example returns an XML document that contains a top-level directory element and a child element named file that represents each XML file found in the specified directory. The file element has a size attribute with a value of the file size in bytes, and the value of the file element is the name of the XML file. For example:

```
<directory>
  <file size="10000">one.xml</file>
  <file size="15550">two.xml</file>
</directory>
```

The resulting XML structure can be used in other XQuery expressions, as shown in the query of this example.

ExternalFunctions

From the ExternalFunctions directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where *xquery_file* is one of the following query files in the ExternalFunctions directory:

- javaFunction.xq invokes a static Java method.
- javaInstanceMethod.xq invokes a Java instance method.
- sqlFunction.xq invokes a SQL function.
- sqlFunctionFromModule_db2luw.xq invokes a SQL function from a module that declares SQL functions for DB2 for Linux/UNIX/Windows.
- sqlFunctionFromModule_oracle.xq invokes a SQL function from a module that declares SQL functions for Oracle.
- sqlFunctionFromModule_sqlserver.xq invokes a SQL function from a module that declares SQL functions for Microsoft SQL Server.
- jdbcEscapeFunction.xq invokes a JDBC escape function.

If you do not specify an XQuery file in the command line, a prompt appears asking you to enter a number that corresponds to the XQuery file you want to execute.

The result of the query is written to standard output.

ExternalVariables

From the ExternalVariables directory, enter the following command line to execute the example:

Windows:

```
run.bat
```

UNIX:

```
run.sh
```

A prompt appears asking you to enter one of the following numbers that correspond to the type of external variable you want to execute:

- 1 - Bind an xs:int external variable
- 2 - Bind an xs:string external variable
- 3 - Bind a DOM node to an external variable
- 4 - Bind an XQItem to an external variable
- 5 - Bind an XQSequence to an external variable

JNDIDataSource

Before using this example, you must have a JNDI provider on your machine. If you do not have one, you can download one from:

<http://javashopl.m.sun.com/ESCom/docs/Welcome.jsp?StoreId=22&PartDetailId=7110-jndi-1.2.1-oth-JPR&SiteId=JSC&TransactionId=noreg>

You must place the `providerutil.jar` and `fscontext.jar` files in your CLASSPATH. You can edit the `setenv.bat` (Windows) or `setenv.sh` (UNIX/Linux) files to do this.

From the `JNDIDataSource` directory, enter the following command line to execute the example:

Windows:

```
run.bat
```

UNIX:

```
run.sh
```

RDBMSUpdate

From the `RDBMSUpdate` directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where `xquery_file` is one of the following query files in the `RDBMSUpdate` directory:

- `insert-holdings.xq` inserts data into the holdings table.
- `update-holdings.xq` updates data in the holdings table.
- `delete-holdings.xq` deletes data from the holdings table.
- `update-holdings-from-xml.xq` uses data provided in an XML document to update data in the holdings table.
- `update-function.xq` updates data in the holdings table using a user-defined function.

- shredding-xml.xq shreds data provided in an XML document into multiple tables.

If you do not specify a query file name, a prompt appears asking you to enter the number that corresponds to the XQuery file you want to use:

```
1 - insert-holdings.xq
2 - update-holdings.xq
3 - delete-holdings.xq
4 - update-holdings-from-xml.xq
5 - update-function.xq
6 - shredding-xml.xq
```

Optionally, you can write your own query, save it to a file, and execute that XQuery file by entering the file name as the argument to run.bat or run.sh.

ResultRetrieval

From the ResultRetrieval directory, enter the following command line to execute the example:

Windows:

```
run.bat
```

UNIX:

```
run.sh
```

A prompt appears asking you to enter one of the following numbers that correspond to the type of retrieval method you want to use:

```
1 - SAX
2 - STAX
3 - DOM
```

The results of the retrieval method are displayed in the standard output.

UpdateFacility

From the UpdateFacility directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where *xquery_file* is one of the following query files in the UpdateFacility directory:

- *change-values.xq* uses the *replace* expression to replace the node value of a queried document .
- *insert-nodes.xq* uses the *insert* expression to insert a new node.
- *rename-nodes.xq* uses the *rename* expression to rename an existing node.
- *transform-change-values.xq* uses *copy*, *modify*, and *return* clauses to replace the value of an existing node.
- *transform-insert-nodes.xq* uses *copy*, *modify*, and *return* clauses to insert a new node.

If you do not specify a query file name, a prompt appears asking you to enter the number that corresponds to the XQuery file you want to use:

```
1 - rename-nodes.xq
2 - change-values.xq
3 - insert-nodes.xq
4 - transform-change-values.xq
5 - transform-insert-nodes.xq
```

Optionally, you can write your own query, save it to a file, and execute that XQuery file by entering the file name as the argument to `run.bat` or `run.sh`.

XMLQuery

From the XMLQuery directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where *xquery_file* is one of the following query files in the XMLQuery directory:

- `query-initial-context.xq` queries the initial context document.
- `query-doc-function.xq` queries an XML document using the `fn:doc()` function.
- `query-external-variable.xq` queries an XML document using an external variable.
- `query-directory.xq` queries multiple XML documents contained in the same directory.
- `query-pipeline-1.xq` uses the result of a one XQuery to provide input into another XQuery.

If you do not specify a query file name, a prompt appears asking you to enter the number that corresponds to the XQuery file you want to use:

```
1 - query-initial-context.xq
2 - query-doc-function.xq
3 - query-external-variable.xq
```



```
4 - query-directory.xq
5 - query-pipeline-1.xq
```

Optionally, you can write your own query, save it to a file, and execute that XQuery file by entering the file name as the argument to `run.bat` or `run.sh`.

XQJExecute

From the XQJExecute directory, enter the following command line to execute the example:

Windows:

```
run.bat [xquery_file]
```

UNIX:

```
run.sh [xquery_file]
```

where *xquery_file* is one of the following query files in the XQJExecute directory:

- `collection-users.xq` returns all data from the users table.
- `collection-holdings.xq` returns all data from the holdings table.
- `flwor.xq` joins data from two relational tables using a nested FLWOR expression.
- `JoinXMLToRelational.xq` joins data from an XML file with data from a relational table.
- `function.xq` uses local function declarations.
- `portfolioHTML.xq` serializes the query result as HTML.
- `MainModule.xq` and `LibraryModule.xq` use XQuery modules.
- `nodeId.xq` uses a path expression to eliminate duplicate XML nodes.

If you do not specify an XQuery file in the command line, a prompt appears asking you to enter a number that corresponds to the XQuery file you want to execute. Or, you can enter the number "9" and then type in the text of your own query, which will be executed.

I Troubleshooting

This appendix provides information about using Stylus Spy *for* XQJ and Java logging to troubleshoot problems, and information about resolving the following types of errors:

- fn:collection errors
- Static type errors

Logging XQJ Calls with Stylus Spy™ *for* XQJ

Stylus Spy passes XQJ calls issued by an application to Stylus XQuery and logs detailed information about those calls, which you can use for troubleshooting.

Stylus Spy logging for connections is not enabled by default. You can enable Stylus Spy logging and configure it for your needs by setting one or multiple options (attributes) for Stylus Spy. For example, you may want to direct logging to a local file on your machine.

When Stylus Spy logging is enabled for a connection, you can turn logging on and off at runtime using the `setEnabledLogging()` method in the `com.ddtek.xquery.xqj.ExtLogControl` interface. See [“Generating a Stylus Spy™ Log” on page 559](#) for information about using a Stylus Spy log for troubleshooting.

Enabling Stylus Spy™ Logging

To enable Stylus Spy logging, set one or multiple Stylus Spy attributes using any of the following methods:

- Using the `SpyAttributes` property of the `DDXQDataSource` class to set Stylus Spy attributes explicitly in your application
- Using the `SpyAttributes` property of the `DDXQDataSource` class to set Stylus Spy attributes in a data source object in JNDI

Setting Stylus Spy Attributes Explicitly in a Java Application

To set Stylus Spy attributes explicitly in your application, use XQJ to construct a `DDXQDataSource` instance in your Java application and specify the `SpyAttributes` property of the `DDXQDataSource` class. The format for the value of the `SpyAttributes` property is:

```
spy_attribute=value[;spy_attribute=value]...
```

where *spy_attribute=value* is a Stylus Spy attribute and a valid value for that attribute. See [“Stylus Spy Attributes” on page 557](#) for a list of supported attributes.

Example on Windows

The following example enables Stylus Spy logging and configures Stylus Spy to log all XQJ activity to a log file, including the content of SAX streams passed through XQJ to the spy.log file located in the C:\temp directory (log=(file)C:\\\\temp\\\\spy.log;logSAX=yes).

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks");
ds.setSpyAttributes("log=(file)C:\\\\temp\\\\spy.log;logSAX=yes");
Context ctx = new InitialContext();
ctx.bind("holdings_ds", ds);
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

NOTE: If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash, as shown in this example.

Example on UNIX/Linux

The following example enables Stylus Spy logging and configures Stylus Spy to log all XQJ activity to a log file located in the /tmp directory (log=(file)/tmp/spy.log;logTimestamp=yes). The spy.log file includes a timestamp on each line in the log.

```
DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:oracle://server1:1521;SID=ORCL");
ds.setSpyAttributes("log=(file)/tmp/spy.log;logTimestamp=yes");
Context ctx = new InitialContext();
ctx.bind("holdings_ds", ds);
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

Setting Stylus Spy Attributes in JNDI

To set Stylus Spy attributes in JNDI, configure a data source object that specifies the `SpyAttributes` property of the `DDXQDataSource` class and use XQJ to load the `DDXQDataSource` object. The format for the value of the `SpyAttributes` property is:

```
(spy_attribute=value[;spy_attribute=value]...)
```

where `spy_attribute=value` is a Stylus Spy attribute and a valid value for that attribute. See [“Stylus Spy Attributes” on page 557](#) for a list of supported attributes.

Example on Windows

The following example enables Stylus Spy logging and configures Stylus Spy to log all XQJ activity to a log file located in the `C:\temp` directory

(`log=(file)C:\\\\temp\\\\spy.log;logTimestamp=yes`). The `spy.log` file includes a timestamp on each line in the log.

```
XQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:oracle://server1:1521;SID=ORCL");
ds.setSpyAttributes("log=(file)C:\\\\temp\\\\spy.log;logSAX=yes");
Context ctx = new InitialContext();
ctx.bind("holdings_ds", ds);
XQConnection conn = ds.getConnection("myuserid","mypswd");
```

NOTE: If coding a path on Windows to the log file in a Java string, the backslash character (`\`) must be preceded by the Java escape character, a backslash, as shown in this example.

Example on UNIX/Linux

The following example enables Stylus Spy logging and configures Stylus Spy to log all XQJ activity to a log file located in the /tmp directory

(log=(file)/tmp/spy.log;logTimestamp=yes). The spy.log file includes a timestamp on each line in the log.

```
XQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:oracle://server1:1521;SID=ORCL");
ds.setSpyAttributes("log=(file)/tmp/spy.log;logTimestamp=yes");
Context ctx = new InitialContext();
ctx.bind("holdings_ds", ds);
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

Stylus Spy Attributes

[Table I-1](#) describes the attributes you can set for Stylus Spy.

Table I-1. Stylus Spy Attributes

Attribute	Description
enable={yes no}	Enables and disables Stylus Spy logging. The default is yes (if any Stylus Spy attributes are set as described in “Enabling Stylus Spy” Logging on page 554 , Stylus Spy logging is enabled). Once enabled, you can turn Stylus Spy logging on and off at runtime using the <code>setEnabledLogging()</code> method in the <code>com.ddtek.xquery.xqj.ExtLogControl</code> interface. See “Turning On and Off Stylus Spy” Logging on page 560 for more information.
linelimit= <i>numberofchars</i>	Sets the maximum number of characters that Stylus Spy will log on any one line. The default is 0 (no maximum limit).

Table I-1. Stylus Spy Attributes *(cont.)*

Attribute	Description
log=System.out	Directs logging to the Java output standard, System.out.
log=(file) <i>filename</i>	Redirects logging to the file specified by <i>filename</i> .
logDOM={yes no}	Specifies whether Stylus Spy logs the content of DOM trees passed through XQJ. The default is no (DOM tree content is not logged). To log the content of DOM trees, set the value of this attribute to yes.
logSAX={yes no}	Specifies whether Stylus Spy logs the content of SAX event streams passed through XQJ. The default is no (SAX event stream content is not logged). To log the content of SAX events, set the value of this attribute to yes.
logStAX={yes no}	Specifies whether Stylus Spy logs the content of StAX event streams passed through XQJ (StAX event stream content is not logged). To log the content of StAX events, set the value of this attribute to yes.
logTName={yes no}	Specifies whether Stylus Spy logs the name of the current thread. The default is no (name of the current thread is not logged). To log the name of the current thread, set the value of this attribute to yes.
logTimestamp={yes no}	Specifies whether a timestamp is logged on each line of the Stylus Spy log. The default is no. To log the timestamp on each line of the log, set the value of this attribute to yes.

Generating a Stylus Spy™ Log

This section provides an example of a typical Stylus Spy log and instructions for turning on and off Stylus Spy logging at runtime.

Stylus Spy™ Log Example

The following example shows a Stylus Spy log. The numbers in superscript are note indicators that correspond to the notes following the example. They provide explanations for the referenced text to help you understand the content of your own Stylus Spy logs.

NOTE: The following example does not show logging of XML results that occur when the logDOM, logSAX, or logStAX attributes are set to yes.

```
spy>> XQConnection(0).createExpression()
spy>> OK (XQExpression(0))1
spy>> XQExpression(0).executeQuery(String query)
spy>> query : 123,'hello world!'
spy>> OK (XQResultSequence(0))2
spy>> XQResultSequence(0).next()
spy>> OK (true)3
spy>> XQResultSequence(0).getObject()
spy>> OK (123)4
spy>> XQResultSequence(0).next()
spy>> OK (true)3
spy>> XQResultSequence(0).getObject()
spy>> OK (hello world!)4
spy>> XQResultSequence(0).next()
spy>> OK (false)5
spy>> XQExpression(0).close()
spy>> OK6
spy>> XQConnection(0).close()
spy>> OK7
```

- 1 An XQExpression is created. Further in the Stylus Spy output, this XQExpression is identified by `XQExpression(0)`.
- 2 The query `123, 'hello world!'` is executed. An `XQResultSequence` is returned.
- 3 The application moved to the next item in the result sequence. `True` is returned, indicating that there is another item in the result sequence.
- 4 The item's data is retrieved into the application.
- 5 The application moved to the next item in the result sequence. `False` is returned, indicating that the end of the sequence has been reached.
- 6 The application closes the `XQExpression`.
- 7 The application closes the `XQConnection`.

Turning On and Off Stylus Spy™ Logging

Once Stylus Spy logging is enabled for a connection, you can turn on and off the logging at runtime using the `setEnableLogging()` method in the `com.ddtek.xquery.xqj.ExtLogControl` interface. When logging is enabled, all `Connection` objects returned to an application provide an implementation of the `ExtLogControl` interface.

For example, the following code turns off logging using `setEnableLogging(false)`:

```
import com.ddtek.xquery.xqj.ExtLogControl

DDXQDataSource ds = new DDXQDataSource();
ds.setJdbcUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=stocks;");
ds.setSpyAttributes("log=(file)/tmp/spy.log");
XQConnection conn = ds.getConnection("myuserid","mypswd");

((ExtLogControl) conn).setEnableLogging(false);
...
```

The `setEnableLogging()` method only turns on and off logging if Stylus Spy logging has already been enabled for a connection; it does not set or change Stylus Spy attributes. See [“Enabling Stylus Spy™ Logging” on page 554](#) for information about enabling and customizing Stylus Spy logging.

ExtLogControl Class

ExtLogControl Class

Methods

Description

`void setEnableLogging(boolean)`

If Stylus Spy was enabled when the connection was created, you can turn on or off Stylus Spy logging at runtime using this method. If true, logging is turned on. If false, logging is turned off. If Stylus Spy logging was not enabled when the connection was created, calling this method has no effect.

`boolean getEnableLogging()`

Indicates whether Stylus Spy logging was enabled when the connection was created and whether logging is turned on. If the returned value is true, logging is turned on. If the returned value is false, logging is turned off.

Java Logging

NOTE: Enable Java logging only when IVI Technologies technical support instructs you to do so.

Stylus XQuery uses standard Java logging as provided in J2SE 1.4.x or later. The simplest way to enable and disable logging is through the `logging.properties` file, which is installed as part of your J2SE installation in the `/jre/lib` subdirectory in your J2SE installation directory.

In the `logging.properties` file, you can specify the level of logging that you want to use, whether you want to write messages as text (simple) or as XML, and whether the messages are written to a console or to a file. Note that the logging level must be set to `FINE` for Stylus XQuery to log messages.

Examples: Modifying `logging.properties`

- Log all messages to standard out using the `SimpleFormatter` and limit all messages that are logged to equal to or lower than `FINE` by:
 - a Modifying the default global logging level to:
`.level = FINE`
 - b Modifying the `java.util.logging.ConsoleHandler.level` line to:
`java.util.logging.ConsoleHandler.level = FINE`
 - c Modifying the `java.util.logging.ConsoleHandler.formatter` line to:
`java.util.logging.ConsoleHandler.formatter =
 java.util.logging.SimpleFormatter`

- Log all messages to standard out using the SimpleFormatter, log only SEVERE messages for all software other than Stylus XQuery, and limit the messages logged for Stylus XQuery to equal to or lower than FINE by:
 - a Modifying the default global logging level to:


```
.level = SEVERE
```
 - b Modifying the facility-specific properties line to:


```
com.ddtek.xquery.level = FINE
```
 - c Modifying the `java.util.logging.ConsoleHandler.level` line to:


```
java.util.logging.ConsoleHandler.level = FINE
```
 - d Modifying the `java.util.logging.ConsoleHandler.formatter` line to:


```
java.util.logging.ConsoleHandler.formatter =  
    java.util.logging.SimpleFormatter
```
- Disable logging by modifying the default global logging level to:


```
.level = CONFIG
```

To enable logging that uses a file other than the default:

- 1 Create a local copy of the `logging.properties` file.
- 2 Modify the file.
- 3 Enable logging by using the following command line:

```
java -Djava.util.logging.config.file =  
    path/name_of_loggingfile
```

Resolving fn:collection Errors

Stylus XQuery uses `fn:collection` to access a relational table or to access multiple XML files in a directory. For example, the following query accesses the holdings database table:

```
collection('holdings')
```

When Stylus XQuery cannot resolve the `fn:collection` argument to a specific database object or to a file system directory, it raises an error.

Guidelines for Resolving Errors

When Stylus XQuery cannot resolve the `fn:collection` argument to a specific database object, it raises an error such as:

```
Table x not found in any JDBC connection or Table x found  
in multiple JDBC connections.
```

If you encounter these types of errors when using Stylus XQuery, the following guidelines will help you troubleshoot and correct the cause of the error:

- Qualify table names in `fn:collection` arguments if you have multiple database tables with the same name or the default catalog and schema associated with the connection do not provide access to the database table.
- Escape special characters in catalog, schema, and table names.
- Verify connections associated with the query.
- When querying XML files in a directory, make sure you specify the directory URL correctly. One typical mistake is that the `file:///` URL prefix was not specified as part of the directory URL.

Qualifying Table Names

If you have multiple database tables with the same name or the default catalog and schema associated with the connection do not provide access to the database table, you can qualify the database table name in the fn:collection argument to target the specific table. For example, if the default catalog (database) and schema (user) associated with the connection is financial and joseph, respectively, and the target table is owned by the schema mary, qualify the table name in the fn:collection argument:

```
collection('financial.mary.holdings')
```

Using Catalog and Schema Names

To verify that you know the correct catalog name, schema name, and table name, start the SQL tool shipped with your database and connect to the database server. Once connected, execute the following SQL statement against the database:

DB2 and Microsoft SQL Server

```
SELECT * FROM "catalog"."schema"."table" WHERE 1=0
```

Informix

```
SELECT * FROM "catalog":"schema"."table" WHERE 1=0
```

MySQL

```
SELECT * FROM 'catalog'.'table' WHERE 1=0
```

Oracle and PostgreSQL

```
SELECT * FROM "schema"."table" WHERE 1=0
```

Sybase

```
SELECT * FROM catalog.schema."table" WHERE 1=0
```

where *catalog*, *schema*, and *table* are the catalog name, schema name, and table name of the database object you are trying to access.

NOTE: Oracle and PostgreSQL databases do not have catalogs. MySQL databases do not have schemas.

If the SQL statement returns an empty result, the values you entered correspond to the correct catalog name, schema name, and table name. Use these values in the `fn:collection` argument to qualify the table name. If the SQL statement returns an error, the values you entered are incorrect.

IMPORTANT: The case of the values specified in the `fn:collection` argument must match the case of the database.

On Microsoft SQL Server and Sybase, a user can have the special status of database owner. For example, if the SQL name of the target table is "financial"."dbo"."holdings," qualify the table name in the `fn:collection` argument with the schema name `dbo`:

```
collection('financial.dbo.holdings')
```

Using JDBC Connection Names

If the table name in the `fn:collection` argument is qualified with a catalog name or schema name (or both) and Stylus XQuery returns an error indicating that multiple tables with the same name exist, you need to qualify the table name using a JDBC connection name. A JDBC connection name identifies a specific connection associated with the database table.

Suppose two tables of the same name, `holdings`, exist on different database servers with the same schema name, `joseph`. In this case, the following query does not provide enough information for Stylus XQuery to locate the target table:

```
collection('joseph.holdings')
```


To identify the correct table, you can qualify the table name in the `fn:collection` argument with a JDBC connection name. Here's an example that shows a connection made explicitly in the application to two different databases; each connection is assigned a unique JDBC connection name, `stocks1` and `stocks2`, respectively:

```
DDXQJDBCCConnection jc1 = new DDXQJDBCCConnection();
jc1.setUrl("jdbc:xquery:sqlserver://server1:1433;databaseName=financial");
jc1.setName("stocks1");
DDXQJDBCCConnection jc2 = new DDXQJDBCCConnection();
jc2.setUrl("jdbc:xquery:oracle://server2:1521;SID=ORCL");
jc2.setName("stocks2");
DDXQDataSource ds = new DDXQDataSource();
ds.setDdxqJdbcConnection(new DDXQJDBCCConnection[] {jc1, jc2});
XQConnection conn = ds.getConnection("myuserid", "mypswd");
```

Here's an example showing the same connection information configured in a Stylus XQuery source configuration file:

```
...
<JDBCCConnection name="stocks1">
  <description>connection to stocks1 data</description>
  <url>jdbc:xquery:sqlserver://localhost:1433;DatabaseName=financial</url>
  <user>myuserid</user>
  <password>mypswd</password>
  ...
<JDBCCConnection name="stocks2">
  <description>connection to stocks2 data</description>
  <url>jdbc:xquery:oracle://localhost:1521;SID=ORCL</url>
  <user>myuserid</user>
  <password>mypswd</password>
  ...
```

To target the holdings table on server1, qualify the table name with the JDBC connection name `stocks1` in addition to the catalog name and schema name:

```
collection('stocks1:joseph.holdings')
```

See [“Choosing a Connection Method” on page 123](#) for more information about connecting with Stylus XQuery.

Escaping Special Characters

If the catalog name, schema name, or table name in the `fn:collection` argument contains a period (`.`), colon (`:`), or backslash (`\`), escape the character with a backslash (`\`) so that Stylus XQuery can parse the argument into its different parts. For example, if the target table is named `a.holdings` and you specify the following query, Stylus XQuery parses `'A'` as the schema name, not as part of the table name:

```
collection('a.holdings')
```

Escaping the period (`.`) in the `fn:collection` argument using the backslash character allows Stylus XQuery to parse the argument correctly:

```
collection('a\.holdings')
```

In addition, XQuery string literal syntax applies to the `fn:collection` argument. If a table name contains double quotes, for example, `a"holdings`, and the `fn:collection` argument uses double quote delimiters, you must repeat the double quotes:

```
collection("a""holdings")
```

Or, you can use:

```
collection('a"holdings')
```

See the next section [“Using Aliases”](#) for details about how to avoid escaping characters.

Using Aliases

To avoid escaping period (`.`), colon (`:`), and double quotes (`"`), and to avoid the SQL/XML escaping of non-supported XML characters,

Stylus XQuery supports an alias attribute for the catalog, schema, and table elements of the source configuration file.

For example, assume a table exists named `tab"le` that contains a single integer column named `c` with one row. In this case, the following query:

```
collection("tab""le")
```

returns:

```
<tab_x0022_le>
  <c>1</c>
</tab_x0022_le>
```

Using the alias attribute for the table element as shown in the following source configuration file example:

```
<catalog name="catalog">
  <schema name="schema">
    <table name='tab"le' alias="tablealias"/>
  </schema>
</catalog>
```

you can specify `tablealias` as the table name in `fn:collection`:

```
collection("tablealias")
```

which results in:

```
<tablealias>
  <c>1</c>
</tablealias/>
```

Verifying Connections

Using Stylus XQuery, an application establishes a connection to the database to execute a query. The application can establish a connection to the database in multiple ways: explicitly specifying connection information in the application, using a data source registered with JNDI, or using a Stylus

XQuery source configuration file. If Stylus XQuery cannot access the database because connection information is specified incorrectly or because the structure of the configuration file is incorrect, it raises an error.

Verifying Connection URLs

Verify that the following information in your connection URL is correct:

- Type of database to which the application is connecting.
- TCP/IP address or host name of the database server to which the application is connecting.
- Number of the TCP/IP port.
- User name used to connect to the database.
- Password used to connect to the database.
- Database-specific connection properties that provide additional connection information. DatabaseName (for DB2 and Microsoft SQL Server) and SID (for Oracle) are commonly used properties. For a list of available database-specific connection properties, see the tables in [“Specifying Connection URIs” on page 141](#).

Checklist

If you encounter an error when using `fn:collection` with Stylus XQuery when accessing a relational table, examine the following checklist to resolve the problem:

- Qualify table names in `fn:collection` arguments if you have multiple database tables with the same name or the default catalog and schema associated with the connection do not provide access to the database table.

- Make sure that you know the correct catalog name, schema name, and table name (including case).
- If you are accessing Microsoft SQL Server or Sybase and the database table is owned by dbo, make sure that you qualify the table name with the schema name dbo. For example:

```
collection('financial.dbo.holdings')
```

- If you qualify the table name with a catalog name or schema name (or both) and Stylus XQuery returns an error indicating that multiple collections are found, you may want to qualify the table name with a JDBC connection name.
- Escape special characters in catalog, schema, and table names. See [“Case Sensitivity” on page 122](#) for details about how to avoid escaping characters.
 - Verify connections associated with the query.
 - Make sure that the information specified in your connection URL is correct including: database type, server name, port, user, password, and any database-specific connection properties.
 - If using a Stylus XQuery source configuration file, make sure that it validates against the source_config.xsd schema shipped with Stylus XQuery. This schema is located in the examples/config subdirectory of your Stylus XQuery installation directory. You can validate the configuration file using a tool such as <oxyen> XML Editor for Eclipse (Stylus XQuery Edition) or Stylus Studio.
 - If using a Stylus XQuery source configuration file, make sure that the values of the catalog, schema, and table elements are correct.

Querying XML Files in a Directory

The `fn:collection` argument value is a URL referencing a directory. The URL must use the `file://` scheme. See [“Querying Multiple Files in a Directory” on page 288](#) for complete details.

Some typical errors made when specifying the directory URL are:

- The specified URL is missing the `file:///` URL prefix. For example, `collection(c:/myDir)` must be `collection(file:///c:/myDir)`.
- A forward slash (/) is missing in the directory URL. For example, `collection(file://c:/myDir)` must be `collection(file:///c:/myDir)`.
- The specified URL does not reference a directory.

In addition, you may receive an error when querying XML files in a directory if:

- Some of the files in the specified directory are not well-formed XML documents. Using the `select` property as part of the specified URL, you can control which files are accessed in the specified folder, for example:

```
collection("file:///c:/myDir?select=*.xml")
```

- The XQuery regular expression you specified is not a valid expression. For example, `collection("file:///c:/myDir?select=*.xml;xquery-regex=yes")` must be `collection("file:///c:/myDir?select=.*%5C.xml$;xquery-regex=yes")`.

See [“Querying Multiple Files in a Directory” on page 288](#) for the `collection` function’s declaration for this feature.

Resolving Static Type Errors

Stylus XQuery implements the Static Typing feature of XQuery, which is pessimistic. This means that Stylus XQuery raises errors if an expression cannot be guaranteed to be typesafe.

Static typing provides the following advantages:

- It provides the information needed to perform better SQL generation.
- It can prevent programming errors because it involves analyzing the query before its actually executed.

The disadvantage of static typing is that you must rewrite queries if they do not provide specific type information for certain expressions. For example, you may need to specify the type of an external variable.

Typically, static type errors occur because more specific type information is needed during static analysis, which occurs before any data is encountered. Errors typically are resolved by adding the missing type declaration or making the declaration more specific. For example:

- An arithmetic or comparison operator can work only on certain data types. Static typing ensures that the operand has the correct type. To ensure that `$j + 1` can execute, the type of `$j` must be known statically. Stylus XQuery raises a type error if this type information is not known statically. The solution is to declare the type of the operand.
- In a path expression, the left part of a step must be a node. For example, in the query `$x/*`, static typing ensures that `$x` is a node. Stylus XQuery raises a type error if this type information is not known statically. The solution is to declare the type of the variable as an element or a document node.


```
declare variable $x external;
$x/*
```

The solution is to declare `$x` to be an element.

```
declare variable $x as element() external;
$x/*
```

In Stylus XQuery, XML bound to an external variable is always untyped. It is often convenient to declare it to be untyped in the external variable declaration, which makes it easier to use in queries. For example, consider the following query.

```
declare variable $y as element() external;
$y/b + 1
```

Static type error. Types 'element(b, xs:untyped)*' and 'xs:integer' are invalid argument types for binary operator '+'.

Static typing cannot look at the actual value to which `$y` will be bound, so it raises an error for the query, because the query does not state that `$y` is an element with untyped content. The solution is to explicitly declare that `$y` is bound to an XML element, and also to state that this element is untyped.

```
declare variable $y as element(*, xs:untyped) external;
$y/b + 1
```

Now, the query executes as expected. Suppose `$y` is bound to an XML document containing the same element used in the preceding query. If the external variable is bound to an untyped XML document, rather than an element, you can declare it like this:

```
declare variable $y as document-node(element(*,
    xs:anyType)) external;
```

Types for Initial Context Items

Just as with external variables, static type errors often are encountered in expressions that use the initial context item if they do not have a specific type declared. Unlike external variables, the type of the initial context item is not specified as part of the XQuery, but through the XQJ API, using the `XQStaticContext` object associated with an XQuery expression.

For example, if you want to bind the external variable `$y` to an untyped XML document, you would do this:

```
declare variable $y as document-node(element(*,
  xs:anyType)) external;
```

If you need to do the same for the initial context item, you would define the type of the initial context item and bind it to a document using XQJ:

```
XQStaticContext context = xqConnection.getStaticContext();
cntxt.setContextItemStaticType(xqConnection.createDocumentElementType
  (xqConnection.createElementType(null, XQItemType.XQBASETYPE_UNTYPED)));
xqExpression = xqConnection.createExpression(cntxt);
xqExpression.bindDocument(XQConstants.CONTEXT_ITEM, new
  FileInputStream("myXMLDocument.xml"));
...
```

Union Types

Sometimes an expression can return more than one type of data, but the query writer knows it will be of a given type. Static typing may need to know the types that are expected. Consider the following expression.

```
(1, 'a')[1] + 2
```

The static typing rules of XQuery do not examine the value of the subscript, so they do not know whether the left operand is an

integer or a string. Therefore, Stylus XQuery raises the following error.

```
Error: [Stylus][XQuery][err:XP0004]Error at line 1, column
15. Static type error. Types '(xs:integer?,xs:string?)'
and 'xs:integer' are invalid argument types for binary
operator '+'.

```

When you know that an expression returns the correct type, often the simplest approach is to use a constructor function or a cast expression to guarantee the correct type.

```
xs:integer((1, 'a')[1]) + 2

```

You also can use `treat as` to tell XQuery which type to expect:

```
((1, 'a')[1] treat as xs:integer) + 2

```

In Stylus XQuery, using `treat as` is generally slower because it is compensated.

Types for Sorting

When sorting data, XQuery must be able to compare all the values it encounters. Because static typing cannot examine the values in an expression, it uses the static type information from the query to guarantee that no runtime type errors will be generated because values of incomparable types are being compared. For example, if the query sorts the holdings table and the shares column is a decimal value, the following query is sorted by comparing decimals:

```
for $h in collection("holdings")//holdings
order by $h/shares
return $h

```

Static analysis also applies to XML documents, but all XML documents are untyped, because Stylus XQuery does not support schema validation. Therefore, the following query is

statically valid, but the shares are sorted as strings, not using their decimal values:

```
for $h in doc("holdings.xml")//holdings
order by $h/shares
return $h
```

In most cases, you would sort based on the numeric value of shares, which is easily done using a constructor function:

```
for $h in doc("holdings.xml")//holdings
order by xs:decimal($h/shares)
return $h
```

Static analysis does not allow sorting unless all types used for sorting are comparable. For example, if you wanted to return the holdings in both the holdings database table and an XML file named holdings.xml, you would have a mixture of types that cannot be compared:

```
for $h in doc("holdings.xml")//holdings
collection("holdings")//holdings
order by $h/shares
return $h
```

```
Error: [Stylus][XQuery][err:XP0004]Error at line 30, column
11. Static type error. Order spec contains invalid comparison
of types 'xs:string' and 'DECIMAL_19_4'.
```

The solution is to use a constructor function to ensure that all values are compared as decimals:

```
for $h in doc("holdings.xml")//holdings
collection("holdings")//holdings
order by xs:decimal($h/shares)
return $h
```

Static Typing Extensions

For static typing analysis, Stylus XQuery follows the rules of the XQuery Formal Semantics with the following exceptions:

- Stylus XQuery does not raise static errors for quantifiers that do not match.
- Stylus XQuery provides more precise typing rules for parent and fn:root.
- The static type of a parameter is determined by the argument expression, not the parameter declaration. The following function executes without raising a static error although types for the function parameters are not declared:

```
declare function local:add($left, $right)
{
  $left + $right
};
local:add(1, 1)
```

- The static type of a function return is determined by the function expression, not the declared type. The following example executes without raising a static error although the function return type is not declared:

```
declare function local:one()
{
  1
};
local:one() + 1
```

- Static typing for constructors use knowledge of the resulting structure as specified by the constructor expression. For example, the expression `<a>1 + 1` succeeds because Stylus XQuery knows statically that the value of the `a` element is an integer. Similarly, `<a>/c` fails with error XPST0005 because Stylus XQuery knows statically that the constructed `<a/>` element does not have a `<c/>` element as a child.

- The XPST0005 error can be disabled by specifying the detect-XPST0005 option declaration.

Index

A

- accessing
 - values of an XQuery item 517
 - XML results of a query 530
- accessors 362
- adding and subtracting durations 379
- advantages, static typing 573
- aggregate functions 386
- aggregating data
 - Streaming XML and 183
- AllowJavaFunctions property 128
- anyURI data type, functions and operators for 372
- applet, Java 2 Platform, permissions for 151, 162
- architecture, Stylus XQuery 55
- arithmetic expressions 350
- assembling strings 369
- atomic types
 - constructor functions 364
 - mapping to XQJ 533
 - supported 459
- authentication
 - client (SSL) 168
 - Kerberos 147
 - NTLM 159
 - server (SSL) 166
 - support for 146
 - types of 146
- AuthenticationMethod property
 - DB2 461
 - SQL Server 468
 - Sybase 481

B

- base URI 129
- base64Binary, functions and operators on 382
- BaseUri property 129
- basics, XQuery 341
- benefits of XQuery 48
- binding sequence
 - definition 93
- books
 - online 29
 - using 26
- boolean constructor functions 373
- boolean values
 - functions on 373
 - operators on 373
- built-in functions
 - ddtek:analyze-edi-from-string() 390
 - ddtek:analyze-edi-from-url() 391
 - ddtek:convert-to-xml() 393
 - ddtek:decimal() 394
 - ddtek:edi-to-xml-from-string() 394
 - ddtek:edi-to-xml-from-url() 396
 - ddtek:format-date 397
 - ddtek:format-date-time 401
 - ddtek:format-number 402
 - ddtek:format-time 405
 - ddtek:http-delete 407
 - ddtek:http-get 408
 - ddtek:http-head 409
 - ddtek:http-options 410
 - ddtek:http-post 411
 - ddtek:http-put 412
 - ddtek:http-trace 414
 - ddtek:info 415
 - ddtek:is-valid() 416

- ddtek:javaCast() 416
- ddtek:parse() 418
- ddtek:serialize-to-url() 420
- ddtek:trim() 417
- ddtek:validate() 424
- ddtek:validate-and-report 427
- ddtek:wscal() 430

C

- case sensitivity 122
- cast to varchar 282
- casting 388
- certificate
 - SSL 166
- Certificate Authority (CA), SSL 167
- choosing a connection method 123
- CLASSPATH, setting 35
- client authentication
 - SSL 168
- Collation property 129
- collation, default 129, 315
- collations
 - comparing strings 314
 - handling Unicode characters 315
 - list of 314
 - specifying 312
 - user-defined 314
 - W3C Unicode Codepoint 314
- collection
 - NULLID (DB2) 464
 - resolving errors 564
 - specifying in a query 118
- collection URI resolver 298
- CollectionURIResolver property 129
- command line utility 40
- comparing strings, collations 314
- comparison expressions 351
- comparison of
 - duration, date, and time values 375
 - numeric values 368
 - strings 369
- comparisons
 - value compared to general 190
- comparisons in where clauses 189
- compensating for functions with no SQL
 - equivalent 191
- component extraction functions 377
- conditional expressions 354
- configuring
 - environment to run Stylus XQuery
 - examples 540
 - Kerberos authentication 149
 - NTLM authentication 160
 - SSL encryption
- configuring connections
 - quick start 36
- XQJ
 - explicitly 123
 - using JNDI 127
- configuring data source connections
 - multiple databases 124
 - single databases 124
- connect descriptor parameters in
 - tnsnames.ora file (Oracle) 478
- connection pooling 193
 - XQueryWebService framework and 214
- connection properties
 - DB2 460
 - Informix 466
 - MySQL Enterprise 470
 - Oracle 471
 - SQL Server 467
 - Sybase 481
- connection URLs 141
- connections
 - choosing a data source connection
 - method 123
 - configuring 36
 - configuring data source connections 115
 - managing client-server connections 235
 - multiple databases 124
 - permissions 162
 - settings for sockets and connections 236
 - single database 124
 - specifying connection URIs for 141

- using DDXQDataSource to connect to a data source 123
- constructing an XQDataSource instance 123
- constructor functions
 - boolean 373
 - for QNames 381
 - for user-defined types 366
 - for XML schema built-in types 364
 - for xs:dateTime 366
- constructors 352
- contacting Technical Support 33
- context functions 387
- controlling precision and scale for
 - xs:decimal() 279
- conventions, typographical 31
- converting
 - EDI to XML 299
- converting EDI to XML
 - built-in function for 393
 - validating EDI streams 390, 391, 394, 396
- cookies
 - managing cookies in Web service applications 242
- CreateDefaultPackage property (DB2) 462
- creating
 - an XDM instance 418
 - DB2 packages 464
 - XML 83
- custom URI resolver 295, 298

D

- data
 - aggregating 183
 - grouping data 92
 - streaming XML data 177, 237
 - updating data sources 101, 112
 - updating relational data sources 267
- data bases
 - using relational data in XQuery 118

- data encryption
 - database-specific encryption 164
 - SSL
 - for DB2 170
 - for Microsoft SQL Server 172
 - for Oracle 171
 - for Sybase 175
 - SSL support 164
 - support for 163
 - supported types of 164
- data model
 - SQL 2003 standard for XML mappings 120
- data model representation
 - relational database tables 120
 - XML documents 117
- data source
 - configuring connections to a single source 124
 - configuring connections to multiple sources 124
- data source connections
 - authentication 145
 - built-in drivers for 141
 - connection URI format 141
 - third-party drivers 143
 - data encryption 163
 - securing 145
- data sources
 - authenticating connections 146
 - configuring connections explicitly 123
 - configuring connections using JNDI 127
 - connection methods for 123
 - relational 118
 - relational data bases 118
 - Streaming XML and 180
 - updating 112
 - updating using XUF 101
 - using data sources in queries 115
 - using in queries 115
 - XML 116
- data types
 - DB2 448
 - Informix 449

- mapping database data types to XML
 - schema data types 447
- MySQL 451
- Oracle 452
- SQL Server 455
- support for XML Type 263
- support for XML Type data 263
- Sybase 457
- XML 117
- database connections
 - XQueryWebService framework example 213
- DatabaseName property
 - DB2 462
 - Informix 466
 - MySQL Enterprise 470
 - SQL Server 468
- databases
 - authenticating connections 146
 - configuring connections to a single source 124
 - configuring connections to multiple sources 124
 - updating 267
- Stylus Spy
 - attributes
 - setting explicitly in a Java application 554
 - setting in JNDI 556
 - generating a log 559
 - log example 559
 - logging
 - content of DOM trees 558
 - content of SAX event streams 558
 - content of StAX event streams 558
 - generating a log 559
 - turning on and off 560
 - XQJ calls to System.out 558
 - setEnabledLogging() method 560
 - setting attributes for using
 - DDXQDataSource 135
 - SpyAttributes property, specifying 554
- Stylus XML Converters
 - using with Stylus XQuery 56
- Stylus XML Converters, serialization support 442
- Stylus XQuery
 - about 25
 - architecture 55
 - atomic types supported 459
 - authentication support 146
 - building a Web service client 229
 - built-in functions 389
 - comments, support of 347
 - configuring connections 36
 - data encryption support 163
 - ddtek:info built-in function 415
 - features 47
 - jar files in CLASSPATH variable 35
 - Kerberos authentication support 147
 - overview 47
 - SAX events 531
 - serializing, text 532
 - SQL adaptor 56
 - StAX events 531
 - threading 530
 - tutorial 59
 - using with Stylus XML Converters 56
 - using with Stylus Studio 57
 - XML adaptor 56
 - XQJ support 507
 - XQuery expressions 347
 - XQueryWebService framework 201
- dates
 - formatting 397, 401
 - formatting examples 400
- date-time
 - formatting examples 402
- dateTime, date, and time
 - adding and subtracting durations 379
 - values, timezone adjustment on 379
- DB2
 - AuthenticationMethod property 461
 - BINDADD privileges 465
 - collection name 463
 - configuring SSL 170
 - connection properties 460
 - CreateDefaultPackage property 462

- creating packages 462
- data encryption for 165
- data types 448
- DatabaseName property 462
- dynamic sections 464
- DynamicSections property 462
- Grantee property 462
- GrantExecute property 462
- InitializationString property 463
- Kerberos authentication
 - permissions 152
 - service principal name 152
- Kerberos authentication support 147
- library name 463
- location 463
- LocationName property 463
- owner 464
- PackageCollection property 463
- PackageOwner property 464
- packages, creating 462, 464
- password 464
- Password property 464
- prepared statements, maximum 462
- ReplacePackage property 464
- replacing packages 464
- schema for DB2 packages 462
- user name 464
- User property 464
- versions supported 118
- XML-typed data examples 484
- DB2 external functions 332
- DB2 for z/OS Unicode database, connecting to 132, 139
- DB2 V9.1
 - XML-typed data examples 488
- ddtek:analyze-edi-from-string() 390
- ddtek:analyze-edi-from-url 391
- ddtek:convert-to-xml 393
- ddtek:decimal 394
- ddtek:edi-to-xml-from-string() 394
- ddtek:edi-to-xml-from-url() 396
- ddtek:is-valid 416
- ddtek:javaCast() 416
- ddtek:parse() 418
- ddtek:serialize() 419
- ddtek:serialize-to-url 420
- ddtek:sql-delete 270, 421
- ddtek:sql-insert 268, 422
- ddtek:sql-update 269, 423
- ddtek:trim() 417
- ddtek:validate() 424
- ddtek:wscall() 430
- DDXQDataSource 129
 - AllowJavaFunctions property 128
 - BaseUri property 129
 - Collation property 129
 - constructing an XQDataSource instance 123
 - DocumentUriResolver property 129
 - JdbcName property 129
 - JdbcOptions property 130
 - JdbcSqlXmlForest property 130
 - JdbcSqlXmlIdentifierEscaping property 131
 - JdbcTempTableColumns property 132
 - JdbcTempTableSuffix property 132
 - JDBCTransactionIsolationLevel property 133
 - JdbcUrl property 134
 - loading object from JNDI 127
 - MaxPooledQueries property 134
 - ModuleUriResolver property 134
 - Password property 135
 - properties 128
 - SpyAttributes property 135
 - User property 135
- DDXQJDBCConnection
 - Name property 136
 - Options property 137
 - Password property 137
 - properties 128
 - SqlXmlIdentifierEscaping property 138
 - TempTableColumns property 139
 - TempTableSuffix property 139
 - Url property 141
 - User property 141
- declaring Java functions 317
- default collation 129, 315

- deferred binding 532
- deleting relational data 270, 421
- detect-XPST0005 option declaration 277
- disassembling strings 369
- document order 346
- documentation, about 29
- DocumentUriResolver property 129
- DOM
 - compliance level 531
 - logging content of using Stylus Spy 558
 - returning results 69
- DOM trees, querying data from 65
- domain controller 159
- durations, dates, and times
 - comparison of values 375
 - functions on 374
 - operators on 374
- durations, ordered subtypes of 375
- DynamicSections property (DB2) 462

E

- EDI
 - analyzing data sources 299
 - converting to XML 299
 - detecting errors in 299
- enabling
 - logging that uses a file other than the default 563
 - Plan Explain 310
- enclosed expressions 83
- encoding
 - encoding XQuery Web service results 238
- encryption
 - configuring SSL
 - for DB2 170
 - for Microsoft SQL Server 172
 - for Oracle 171
 - for Sybase 175
 - data encryption 163
 - data encryption for DB2 165
 - database-specific 164
 - performance optimization 163
 - SSL support 164
 - support 164
 - supported types of 164
- equality of strings 369
- equals 385
- err:XPST0005, raising during static analysis 277
- error function 363
- error handling
 - static type errors 573
 - support 345
- errors, fn:collection() 564
- escaping of identifiers 131, 138
- evaluate-in-memory extension expression 287
- events, StAX 531
- examples
 - combining data from XML and relational sources 51
 - configuring 540
 - connecting to multiple databases 124
 - creating a specific XML structure 51
 - Stylus Spy, log 559
 - execution plan 311
 - Java example using XQJ 54
 - modifying logging.properties 562
 - net service name entry in tnsnames.ora file 476
 - obtaining a javax.security.auth.Subject for authentication 156
 - provided with Stylus XQuery about 541
 - required software 539
 - simple query using a FLWOR expression 50
 - using the XMLFOREST variable 121
- except 385
- executing updates in XQJ 74, 108
- execution plans
 - example 311
 - format of 307
 - generating 307
- expression context 342

- expression, prepared 522
- expressions
 - arithmetic 350
 - comparison 351
 - conditional 354
 - constructors 352
 - extension expressions 275
 - FLWOR 353
 - introduction 347
 - logical 351
 - on SequenceTypes 355
 - ordered 353
 - path 349
 - primary 348
 - quantified 354
 - sequence 350
 - unordered 353
 - validate 356
- extension expressions
 - description 275
 - evaluate-in-memory 287
 - using 275, 287
- extensions, static typing 579
- external functions
 - Java 317
 - SQL 329
 - supported by Stylus XQuery 315
- external variables
 - types 574
 - XQJ 117
- ExtLogControl class 561

F

- features, Stylus XQuery 47 files
 - querying multiple 290
 - querying multiple files 288
- finding XML nodes 76
- FLWOR expression
 - binding sequence 93
 - definition of 75

- example 50
- grouping data 92
- restructuring data 84
- sliding windows 94
- support for 353
- tumbling windows 94
- window clause 93
- XMLFOREST 121
- fn:collection() errors 564
- format, connection URI
 - third-party drivers 143
- format, execution plans 307
- formatting dates
 - examples 400
- formatting date-time
 - examples 402
- formatting methods
 - ddtek:format-date built-in function 397
 - ddtek:format-date-time built-in function 401
 - ddtek:format-number built-in function 402
 - ddtek:format-time built-in function 405
- formatting numbers
 - examples 404
- formatting time
 - examples 406
- function calls
 - resolving Java function calls 324
- functions
 - aggregate 386
 - aggregation and streaming XML 183
 - based on substring matching 371
 - compensating for functions with no SQL equivalent 191
 - component extraction 377
 - constructor 364
 - external functions 315
 - for anyURI 372
 - introduction 361
 - Java external functions 317
 - on base64Binary and hexBinary 382
 - on boolean values 374
 - on durations, dates, and times 374

- on nodes 383
- on NOTATION 383
- on numeric 367
- on numeric values 368
- on sequences 384
- on string values 370
- on strings 369
- related to QNames 381
- SQL external functions 329
- SQL table functions 337
- that generate sequences 386
- to assemble and disassemble strings 369
- trace 363

G

- general comparisons
 - compared to value comparisons 190
- generated SQL, simplifying 253
- generating
 - Stylus Spy log 559 XQuery
 - execution plan 307
- getting started
 - configuring connections 36
 - with Stylus XQuery after installation 35
 - with Stylus XQuery examples 45 XQJ 38
- global option declaration 134, 276
- Grantee property (DB2) 462
- GrantExecute property (DB2) 462
- grouping data 92

H

- handling Unicode characters for collations 315
- handshake, SSL 166
- help, online 29
- hexBinary, functions and operators on 382

- HTTP functions
 - Stylus XQuery implementations of 231
 - settings for connections and sockets 236
 - XML Schema for requests 247
 - XML Schema for responses 249
- HTTP GET
 - XQueryWebService framework and 203
- HTTP methods
 - ddtek:http-delete built-in function 407
 - ddtek:http-get built-in function 408
 - ddtek:http-head built-in function 409
 - ddtek:http-options built-in function 410
 - ddtek:http-post built-in function 411
 - ddtek:http-put built-in function 412
 - ddtek:http-trace built-in function 414
- HTTP POST
 - XQueryWebService framework and 203

I

- identifiers, escaping of 131, 138
- ignore-whitespace option declaration 277
- Index 581
- Informix
 - connection properties 466
 - data types 449
 - DatabaseName property 466
 - InformixServer property 466
 - InitializationString property 466
 - password 467
 - Password property 467
 - user name 467
 - User property 467
 - versions supported 118, 446
- InformixServer property (Informix) 466
- Infoset mapping 117, 359
- initial naming context 127
- InitializationString property
 - DB2 463
 - Informix 466
 - MySQL Enterprise 471

- Oracle 473
- SQL Server 469
- Sybase 481
- initializing JNDI environment 127
- insert expression 106
- inserting relational data 268, 422
- instance method, Java functions 319
- interface
 - XQConnection 508
 - XQDataSource 512
 - XQDynamicContext 514
 - XQExpression 516
 - XQItem 517
 - XQItemAccessor 517
 - XQItemType 519
 - XQMetaData 521
 - XQPreparedExpression 522
 - XQResultItem 523
 - XQResultSequence 524
 - XQSequence 524
 - XQSequenceType 526
 - XQStaticContext 526
- intersection 385
- invoking a Web service operation 430
- isolation levels 133

J

- JAR files, querying 292
- jar files, setting in CLASSPATH 35
- Java
 - built-in function to return Java system properties 415
 - developing an application that executes a query 38
 - logging 562
 - specifying connection information in the application 123
- Java 2 Platform
 - applet permissions 162
 - permissions
 - for establishing connections 162
 - Kerberos authentication 151
 - Security Manager 151, 162
- Java functions
 - declaring 317
 - disabling 128, 328
 - instance method 319
 - mapping types to XQuery 320
 - resolving the function call 324
 - SequenceType 320
 - static method 318
 - using 317
- Java instance methods
 - notes about using 326
- Java Keystore (JKS)
 - SSL authentication and 168
- Java locale, collations 314
- Java static methods 318
- Java system properties
 - built-in function for 415
- javax.security.auth.Subject 156
- JDBC connection name 129
 - DDXQDataSource property 129
 - DDXQJDBCConnection property 136
 - qualifying table name with 119
- JDBC scalar functions 336
- JDBC URL 134, 141
- JdbcName property 129
- JdbcOptions property 130
- JdbcPragmas property 130
- JdbcSqlXmlForest property 130
- JdbcSqlXmlIdentifierEscaping property 131
- JdbcTempTableColumns property 132
- JdbcTempTableSuffix property 132
- JDBCTransactionIsolationLevel property 133
- JdbcUrl property 134
- JNDI
 - advantages of 123
 - initializing environment 127
 - loading a DDXQDataSource object from 127
 - registering data source object 127
 - using initial naming context to find name of data source object 127

- using JNDIDataSource example file as a template 127
- JNDIDataSource example file
 - using as a template to create data source object for JNDI 127
- joining data 67
- JSR 255 53

K

- Kerberos
 - authentication 147
- Kerberos authentication 158
 - configuring 149
 - database support 147
 - Java 2 platform permissions 151
 - javax.security.auth.Subject 156
 - Kerberos configuration file 149
 - Kerberos server 149
 - Key Distribution Center (KDC) 149
 - kinit command (UNIX and Linux) 158
 - MIT Kerberos 149
 - permissions
 - DB2 152
 - Kerberos 151
 - Oracle 153
 - SQL Server 154
 - Sybase 155
 - requirements 148
 - service principal name
 - DB2 152
 - Sybase 155
 - specifying user credentials 156
 - Ticket Granting Ticket (TGT) 158
 - user credentials, specifying 156
- Kerberos configuration file 149
- Kerberos Key Distribution Center (KDC) 150
- Kerberos realm
 - DB2 152
 - Oracle 153
 - SQL Server 154
 - Sybase 155

- keystore, SSL 168
- kinit command, Kerberos authentication 158

L

- large XML documents, querying 177
- literal expressions 348
- literal translation 281
- literal XML constructors 83
- loading a DDXQDataSource object from JNDI 127
- LoadLibraryPath property
 - Oracle 473
 - SQL Server 469
- LocationName property (DB2) 463
- log for Stylus Spy, generating 559
- logging
 - Stylus Spy 553
 - Java 562
 - using a file other than the default 563
- logging.properties file 562
- logging.properties, modifying 562
- logical expressions 351

M

- mapping
 - a function call to a Java method 416
 - data types between XQuery and Java for
 - Java functions 320
 - Oracle connection properties to
 - tnsnames.ora 478
- matching
 - pattern 372
 - substring 371
- maximum number of prepared statements (DB2) 462
- maximum number of queries in the pool 134
- MaxPooledQueries property 134
- Mediator, Stylus XQuery 55

- method, serialization
 - Stylus XML Converters, using 442
 - standard 440
- methods
 - using Java instance methods 326
- methods, ExtLogControl class 561
- Microsoft SQL Server
 - configuring SSL 172
 - Kerberos authentication support 147
- MIT Kerberos 149
- modules 357
- ModuleUriResolver property 134
- MS Office Open documents
 - querying 292
- multiple databases, configuring connections to 124
- multiple files, querying 288, 290
- multi-threading 530
- MySQL Enterprise
 - connection properties 470
 - DatabaseName property 470
 - InitializationString property 471
 - password 471
 - Password property 471
 - user name 471
 - User property 471
 - versions supported 118, 446
- MySQL, data types 451

N

- Name property 136
- name, JDBC connection 129
- named instances (SQL Server)
 - permissions required for 163
- namespaces, reserved 360
- nodes
 - inserting node values 106
 - renaming node values 108
 - replacing node values 105
- nodes, functions and operators on 383
- NOTATION, functions and operators on 383

- NTLM authentication
 - configuring 160
 - DLL 161
 - requirements 159
- numbers
 - formatting 402
 - formatting examples 404
- numeric values
 - comparison of 368
 - functions on 368
 - operators on 367
- numerics, functions and operators on 367

O

- object, ExtConnection 561
- obtaining a Ticket Granting Ticket 158
- OpenDocument Format documents
 - querying 292
- operators
 - for anyURI 372
 - introduction 361
 - on base64Binary and hexBinary 382
 - on boolean values 373
 - on durations, dates, and times 374
 - on nodes 383
 - on NOTATION 383
 - on numeric values 367
 - on numerics 367
 - related to QNames 381
 - that generate sequences 386
- option declarations
 - connection-specific 275
 - description 275
 - detect-XPST0005 277
 - global 134, 275, 276
 - ignore-whitespace 277
 - plan-explain 277
 - serialize 277
 - specifying
 - methods of 284
 - using DDXQDataSource 130

- using DDXQJDBCCConnection 137
- sql-decimal-cast 279
- sql-extra-checks-trailing-spaces 279
- sql-ignore-trailing-spaces 279
- sql-ora10-use-binary-float-double 283
- sql-order-by-on-values 280
- sql-rewrite-algorithm 280
- sql-rewrite-exists-into-count 283
- sql-simple-convert-functions 280
- sql-simple-string-functions 281
- sql-sybase-temptable-index 284
- sql-sybase-use-derived-tables 284
- sql-unicode-literals 281
- sql-unicode-strings 281
- sql-varchar-cast 282
- using 275
- xml-streaming 278
- optional XQuery features 358
- optional XUF features
 - support for 506
- Options property 137
- ORACLE
 - XML-typed data examples 490
- Oracle
 - configuring
 - tnsnames.ora file 478
 - to retrieve connection information from
 - tnsnames.ora file 477
 - configuring SSL 171
 - connect descriptors in tnsnames.ora files 476
 - connection information, retrieving from
 - tnsnames.ora files 476
 - connection properties 471
 - data types 452
 - InitializationString property 473
 - Kerberos authentication support 147
 - Kerberos authentication, permissions
 - required for 153
 - net service name entries in tnsnames.ora
 - files 476
 - password 474
 - Password property 474

- preventing conflicts when using a
 - tnsnames.ora file 478
- Real Application Clusters (RAC) 474
- SID property 474
- System Identifier (SID) 474
- tnsnames.ora file
 - configuring file 478
 - configuring to reference 477
 - connect descriptor parameters 478
 - connection property mappings to 478
 - retrieving connection information from
 - 475
 - using 476
- TNSNamesFile property 475
- TNSServerName property 475
- User property 475
- versions supported 446
- ORACLE 10g
 - XML-typed data examples 495
- Oracle 10g, using BINARY_FLOAT and BINARY_DOUBLE data types 283
- Oracle external functions 334
- ordered expressions 353
- ordered subtypes of duration 375
- overview
 - Stylus XQuery 47
 - XQJ 53
 - XQuery 48

P

- PackageCollection property (DB2) 463
- PackageOwner property (DB2) 464
- packages, creating (DB2) 464
- parameterizing XML views 120
- Password property
 - DB2 464
- DDXQDataSource 135
- DDXQJDBCCConnection 137
- Informix 467
- MySQL Enterprise 471
- Oracle 474

- SQL Server 470
- Sybase 482
- path expressions
 - for relational sources 81
 - for XML sources 76
 - support for 349
- pattern matching on strings, functions 372
- performance
 - improving XQuery performance 177
- performance considerations
 - SQL generation algorithms 261
 - understanding compensation 191
 - using comparisons 189
 - using query pooling 192
- performance optimization
 - encryption 163
- permissions
 - for establishing connections 162
 - Kerberos authentication 151
 - named instances (SQL Server) 163
- PKCS #12
 - SSL authentication and 168
- Plan Explain 307
 - about 307
 - enabling 310
 - Streaming XML and 181
- plan-explain option declaration 277
- pooling connections 193
- pooling, queries 134, 192
- PostgreSQL
 - versions supported 446
- PostgreSQL JDBC driver 143, 446
- PostgreSQL, versions supported 118
- precision 394
- precision and scale 348
- precision for `xs:decimal()`, controlling 279
- predefined namespaces
 - not reserved 360, 437
 - reserved 438
- predicates
 - defined 79
 - numeric 80
- prepared expression 522
- preparing XQuery statements 72

- preventing conflicts when using a
 - `tnsnames.ora` file 478
- primary expressions 348
- processing
 - Plan Explain feature and 181
- processing model 345
- processing XQuery
 - Streaming XML and 178
- prologs 357
- properties
 - DDXQDataSource 128
 - DDXQJDBCConnection 136
- proprietary functions
 - `ddtek:serialize()` 419
- PSVI mapping 359

Q

- QNames
 - constructor functions 381
 - functions and operators related to 381
 - functions related to 381
- qualifying, table name with JDBC connection
 - name 119
- quantified expressions 354
- queries
 - data sources for 115
- query
 - accessing XML results 530
 - developing a Java application that
 - executes a query 38
 - making queries available as Web services
 - 201
 - syntax
 - for extension expression 284
 - for option declaration 284
- query optimization 132, 139
- query plan 307
- query pooling
 - description 192
 - effect on performance 192

- specifying number of queries in the pool 134
- XQJ and 192
- query results
 - transforming 109
- querying
 - data from DOM trees 65
 - data from XML files 65
 - large XML documents 177
 - multiple files in a directory 288, 290
- querying XML
 - files archived in a ZIP or JAR file 292
 - MS Office Open documents 292
 - OpenDocument Format documents 292
- quick start 35

R

- Real Application Clusters (RAC) (Oracle) 474
- registering data source object with JNDI 127
- relational data
 - data source for queries 118
 - executing XQuery against 251
 - processing with XML 91
 - querying 251
 - XML views and 120
- relational data sources
 - supported databases 118
- relational database tables
 - data model representation 120
 - specifying in a query 118
- relational databases
 - deleting data 270, 421
 - inserting data 268, 422
 - updating 267
 - updating data 269, 423
- relative URIs, resolving 129
- rename expression 108
- replace expression 105
- ReplacePackage property (DB2) 464
- required software for examples provided
 - with Stylus XQuery 539

- reserved namespaces 360
- resolving
 - fn:collection() errors 564
 - static type errors 573
 - URIs 129, 134
- REST
 - Web services and 216
 - XQueryWebService framework and 203
 - XQueryWebService framework example 217
- restructuring data 84
- results
 - transforming query results 109
 - writing query results to I/O 420
- returning results with Java XML APIs 69
- reusing connections 193

S

- SAX
 - logging content of using Stylus Spy 558
 - returning results 69
 - serializing result sequence in events 531
- scale 394
- scale for xs:decimal(), controlling 279
- schema for DB2 packages 462
- Secure Sockets Layer (SSL)
 - See SSL
- security
 - authentication for data source
 - connections 145
 - data encryption for data source
 - connections 163
 - supported methods for securing data
 - source connections 145
- Security Manager for Java 2 Platform 151, 162
- sequence expressions 350
- sequence of items 524
- sequences, functions and operators 384, 386
- SequenceType for Java functions 320

- SequenceTypes, expressions on 355
- serialization support
 - Stylus XML Converters, using for 442
 - standard 440
- serialize option declaration 277
- serializing query results 419
- server authentication, SSL 166
- service principal name for Kerberos
 - authentication
 - DB2 152
 - Sybase 155
- ServicePrincipalName property (Sybase) 482
- setEnabledLogging(), using to turn on and off
 - Stylus Spy logging 560
- setting, CLASSPATH 35
- SID property (Oracle) 474
- sliding windows
 - definition 94
- SOAP 49, 51
 - Web services and 216
 - XQueryWebService framework and 203
 - XQueryWebService framework example 218
- sockets
 - managing client-server connections 235
 - settings for sockets and connections 236
- sorting static typing 577
- specifying
 - collations 312
 - connection URLs 141
 - option declarations 284
 - precision and scale 394
 - URI resolver 294
- SpyAttributes property 135
- SQL
 - comparison to XQuery 48
 - compensating for functions with no SQL equivalent 191
 - translating XQuery into SQL 251
 - XQuery functionality that cannot be directly translated into 191
- SQL 2003
 - data model for XML mappings 120
- SQL adaptor, Stylus XQuery 56
- SQL functions
 - DB2 examples 332
 - JDBC scalar functions 336
 - Oracle examples 334
 - requirements and restrictions 329
 - SQL Server examples 330
 - table functions 337
 - user-defined 335
 - using 329
- SQL generation algorithms 260
- SQL Server
 - AuthenticationMethod property 468
 - connection properties 467
 - LoadLibraryPath 469
 - data types 455
 - DatabaseName property 468
 - InitializationString property 469
 - Kerberos authentication, permissions required for 154
 - named instances, permissions required for 163
 - password 470
 - Password property 470
 - user name 470
 - User property 470
 - versions supported 118, 446
 - XML-typed data examples 497
- SQL Server external functions 330
- SQL table functions 337
- SQL/XML
 - mappings 120
 - parameterizing views 120
 - views 120
- sql-decimal-cast option declaration 279
- sql-extra-checks-trailing-spaces option declaration 279
- sql-ignore-trailing-spaces option declaration 279
- sql-ora10-use-binary-float-double option declaration 283
- sql-order-by-on-values option declaration 280
- sql-rewrite-algorithm option declaration 280

- sql-rewrite-exists-into-count option
 - declaration 283
- sql-simple-convert-functions option
 - declaration 280
- sql-simple-string-functions option
 - declaration 281
- sql-sybase-temptable-index option
 - declaration 284
- sql-sybase-use-derived-tables declaration
 - option 284
- sql-unicode-literals option declaration 281
- sql-unicode-strings option declaration 281
- sql-varchar-cast option declaration 282
- SqlXmlIdentifierEscaping property 138
- SSL
 - about 166
 - certificate 166
 - Certificate Authority (CA) 167
 - client authentication 168
 - configuring
 - for DB2 170
 - for Microsoft SQL Server 172
 - for Oracle 171
 - for Sybase 175
 - handshake 166
 - Java Keystore (JKS) 168
 - keystore 168
 - PKCS #12 Keystore 168
 - server authentication 166
 - support for 164
 - truststore 167
- static context of a query 526
- static method, Java function 318
- static typing
 - advantages 573
 - extensions 579
 - external variables 574
 - overview 573
 - sorting 577
 - union types 576
- StAX
 - event 531
 - logging content of using Stylus Spy 558
 - returning results 69
- Streaming XML
 - aggregating data and 183
 - examples
 - expression with attribute predicate 186
 - path expression with predicate 186
 - simple path expressions 185
 - two XML documents 187
 - XQuery expression with exists 187
 - XQuery expression with fn:data 186
 - XQuery expression with function on Node 186
 - examples of 185
 - illustrating with Plan Explain 181
 - when it is not used 188
 - XML headers and 182
- Streaming XML feature
 - data sources for 180
 - description 178
- strings
 - assembling and disassembling 369
 - comparison 369
 - equality 369
 - functions
 - that use pattern matching 372
 - to assemble and disassemble 369
 - functions on values 370
- Stylus Studio
 - using with Stylus XQuery 57 substring matching, functions based on 371 support authentication 146 encryption 164
- SupportLink 33
- Sybase
 - AuthenticationMethod property 481
 - configuring SSL 175
 - connection properties 481
 - data types 457
 - InitializationString property 481
 - Kerberos authentication
 - permissions 155
 - service principal name 155
 - Kerberos authentication support 147

- password 482
- Password property 482
- ServicePrincipalName property 482
- sql-sybase-temptable-index option
 - declaration 284
- user name 482
- User property 482
- versions supported 118, 446
- syntax for
 - extension expression 284
 - option declaration 284
- system information
 - ddtek:info built-in function 415
- System.out, logging XQJ calls to using Stylus Spy 558

T

- Technical Support, contacting 33
- template for creating data source objects for JNDI 127
- temporary tables 132, 139
- TempTableColumns property 139
- TempTableSuffix property 139
- testing
 - Web service operations 219
- testing queries 40
- text, serializing a result sequence into 532
- threading 530
- Ticket Granting Ticket 158
- time
 - formatting 401, 405
 - formatting examples 406
- timezone adjustment 379
- tnsnames.ora file
 - configuring
 - options for 478
 - to reference file 477
 - connect descriptor parameters 478
 - connection information, retrieving 476
 - mapping Oracle connection properties to 478
 - retrieving connection information from 475
- TNSNamesFile property (Oracle) 475
- TNSServerName property (Oracle) 475
- trace function 363
- transaction isolation levels 133
- transforming query results 109
- translation, literal 281
- trimming whitespaces 417
- truststore file, SSL 167
- tumbling windows
 - definition 94
- turning on and off Stylus Spy logging 560
- tutorial
 - Stylus XQuery
 - configuring connections to relational databases 61
 - executing queries 62
 - joining data from XML and relational sources 67
 - preparing XQuery statements 72
 - querying data from XML files or DOM trees 65
 - returning results as DOM tree 70
 - returning results as SAX event stream 71
 - returning results as StAX event stream 71
 - returning results with Java XML APIs 69
 - XQuery
 - creating XML 83
 - finding XML nodes, path expressions 76
 - FLWOR expressions 84, 92
 - grouping data 92
 - processing XML and relational together 91
 - restructuring data 84
 - types
 - for external variables 574
 - for XQuery 346

U

- Unicode
 - characters, handling for collations 315
 - setting the default encoding for string literals to 348
- union 385
- union types 576
- unordered expressions 353
- updating data sources 112
- updating relational data 74
 - using built-in functions 268
 - using XUF 102
- URI resolver
 - collection 298
 - specifying 294
- URI schemes 116
- URIs
 - resolving 129, 134
 - specifying for XML data source connections 117
- Url property 141
- user credentials, specifying with Kerberos authentication 156
- user name, database 135, 141
- User property
 - DB2 464
 - DDXQDataSource 135
 - DDXQJDBCConnection 141
 - Informix 467
 - MySQL Enterprise 471
 - Oracle 475
 - SQL Server 470
 - Sybase 482
- user-defined collations 314
- user-defined functions 335
- using
 - book 26
 - extension expressions 275, 285
 - initial naming context to find name of data source object registered with JNDI 127
 - Java functions 317

- option declarations 275
- Oracle tnsnames.ora files 476
- SQL functions 329
- utility for testing queries 40

V

- validate expressions 356
- validating query results 427
- validating XML results 416
- value comparisons
 - compared to general comparisons 190
- versions
 - specifying the XQuery version 60
 - XQuery versions supported by Stylus XQuery 60

W

- W3C Unicode Codepoint collation 314
- Web service applications
 - building a Web service client 229
 - connection authentication 233
 - data streaming and 237
 - HTTP functions for Web service clients 231
 - managing connections and sockets 235
 - managing cookies 242
 - response encodings 238
 - settings for connections and sockets 236
 - supported authentication methods 233
 - supported encryption methods 235
 - XML Schema for HTTP requests 247
 - XML Schema for HTTP responses 249
- Web service client
 - definition 229
 - response encodings 238
- Web Service Description Language (WSDL) 221
 - See also* WSDL
- Web service operations

- invoking 430
- testing 219
- Web services
 - building a Web service client 229
 - HTTP functions for 231
 - REST interface 216
 - SOAP interface 216
 - Web service client 229
 - WSDL service references 223
 - XQueryWebService framework 201
- whitespace
 - option for ignoring 277
- whitespaces, trimming 417
- window clause
 - definition 93
- Windows Active Directory
 - Kerberos authentication and 149
- Windows Domain
 - DB2 152
 - Oracle 153
 - SQL Server 154
 - Sybase 155
- writing results to I/O 420
- WSDL
 - generating 221
 - using WSDL service references 223

X

- XML
 - case sensitivity of element and attribute names 122
 - constructors 83
 - data model representation 117
 - data source for queries 116
 - data type 117
 - DOM compliance 531
 - parameterizing views 120
 - processing with relational data 91
 - schema data types 447
 - specifying XML documents as query sources 116

- stored in relational database using an XML data type 117
- streaming XML data 177
- text files/streams 116
- validating 427
- validating query results 416
- XML adaptor, Stylus XQuery 56
- XML constructors 83
- XML constructors, literal 83
- XML data
 - streaming 177, 237
- XML documents, querying 177
- XML files
 - querying in a ZIP or JAR file 292
- XML headers
 - Streaming XML and 182
- XML nodes, finding using path expressions 76
- XML Schema
 - for HTTP requests 247
 - for HTTP responses 249
 - validating query results 427
- XML Type data
 - support for 263
- XML views
 - parameterizing 120
 - relational data and 120
- XMLFOREST variable
 - default 120
 - parameterizing views 120
 - setting 120
- xml-streaming option declaration 278
- XML-typed data
 - DB2 examples 484
 - DB2 V9.1 examples 488
 - ORACLE 10g examples 495
 - ORACLE examples 490
 - SQL Server examples 497
- XPath 1.0 and XQuery 2.0 Data Model specification 117
- XQConnection interface 508
- XQDataSource interface 512
- XQDynamicContext interface 514
- XQExpression interface 516

- XQItem interface 517
- XQItemAccessor interface 517
- XQItemType interface 519
- XQJ
 - Stylus XQuery support for classes, interfaces, and methods 507
 - example, executing an XQuery query 54
 - examples 38
 - executing updates 108
 - external variables 117
 - getting started 38
 - JSR 225 53
 - mapping using get methods 533
 - overview 53
 - query pooling and 192
 - using to configure data source connections explicitly 123
 - using to configure data source connections using JNDI 127
- XQMetaData interface 521
- XQPreparedExpression interface 522
- XQResultItem interface 523
- XQResultSequence interface 524
- XQSequence interface 524, 532
- XQSequenceType interface 526
- XQStaticContext interface 526
- XQuery
 - accessing XML results of a query 530
 - accessors for functions and operators 362
 - adding and subtracting durations 379
 - aggregate functions 386
 - atomic types 459, 533
 - basics 341
 - benefits 48
 - boolean constructor functions 373
 - building a Web service client 229
 - comparison of
 - duration, date, and time values 375
 - numeric values 368
 - strings 369
 - component extraction functions 377
 - concepts supported by Stylus XQuery 346
 - Conformance, Data Model 359
 - constructor functions 364
 - for QNames 381
 - for user-defined types 366
 - for XML schema built-in types 364
 - context functions 387
 - equality of strings 369
 - equals 385
 - error function 363
 - error handling 345
 - example
 - combining data from XML and relational sources 51
 - creating a specific XML structure 51
 - simple query using a FLWOR expression 50
 - except 385
 - executing against relational data 251
 - expression context 342
 - expressions
 - arithmetic 350
 - comparison 351
 - conditional 354
 - constructors 352
 - FLWOR 353
 - logical 351
 - on SequenceTypes 355
 - path 349
 - primary 348
 - quantified 354
 - sequence 350
 - unordered 353
 - validate 356
 - extension expressions 275
 - extensions of expressions to support XUF 505
 - extensions that support XUF 502
 - extensions to built-in functions library to support XUF 505
 - functionality that cannot be directly translated into SQL 191
 - functions
 - based on substring matching 371
 - for anyURI 372
 - introduction 361

- on base64Binary and hexBinary 382
- on boolean values 373, 374
- on durations, dates, and times 374
- on nodes 383
- on NOTATION 383
- on numeric values 368
- on numerics 367
- on sequences 384
- on string values 370
- on strings 369
- related to QNames 381
- that generate sequences 386
- to assemble and disassemble strings 369
- improving performance 177
- intersection 385
- making queries available as Web services 201
- modules 357
- operators
 - for anyURI 372
 - introduction 361
 - on base64Binary and hexBinary 382
 - on boolean values 373
 - on durations, dates, and times 374
 - on nodes 383
 - on NOTATION 383
 - on numeric values 367
 - on numerics 367
 - on sequences 384
 - related to QNames 381
 - that generate sequences 386
- option declarations 275
- optional features 358
- ordered subtypes of duration 375
- overview 48
- predicates 79
- processing model 345
- prologs 357
- query pooling 192
- static typing
 - external variables 574
 - overview 573
 - union types 576
- Streaming XML feature 178

- string functions that use pattern matching 372
- timezone adjustment 379
- trace function 363
- types 346
- union 385
- updating data sources 101
- version support 60
- XML constructors 83
- XQuery Update Facility (XUF) 101
- XQueryWebService framework 201
- XQuery 1.1
 - binding sequence 93
 - sliding windows 94
 - tumbling windows 94
 - window clause 93
- XQuery execution plan 307
- XQuery expressions, not translated to SQL 287
- XQuery processing
 - Plan Explain 181
- XQuery statements, preparing 72
- XQueryWebService framework
 - architecture 204
 - database connections and 213
 - example 211
 - generating WSDL for 221
 - HTTP GET 203
 - HTTP POST 203
 - overview 201
 - REST 203, 216
 - SOAP 203, 216
 - testing 219
 - Web service interfaces 216
 - WSDL service references and 223
- XUF
 - copy clause 109
 - examples
 - inserting node values 107
 - overview 103
 - renaming node values 108
 - replacing node values 105
 - expressions 102
 - insert expression 106

- inserting node values 106
- modify clause 109
- rename expression 108
- renaming node values 108
- replace expression 105
- replacing node values 105
- return clause 109
- support for 102
- transforming query results 109
- updating data sources 112
- XUF support
 - conformance to specification 506
 - extensions of existing XQuery expressions 505
 - extensions to the XQuery processing mode 502
 - extensions to the XQuery static context 503
 - extensions to XQuery 1.0 502
 - extensions to XQuery built-in function library 505
 - new kinds of expressions 504
 - optional features 506
- XUF support
 - extensions to the XQuery prolog 503

Z

- ZIP files
 - creating with `ddtek:serialize-to-url()` 420, 421
 - updating files in with `ddtek:serialize-to-url()` 421
- ZIP files, querying 292