



Component Library Reference



VERSION 3.5

Borland®
JDataStore™

Inprise Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249

Refer to the file DEPLOY.TXT located in the root directory of your JBuilder or JDataStore product for a complete list of files that you can distribute in accordance with the JBuilder or JDataStore License Statement and Limited Warranty.

Inprise may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1997, 1999 Inprise Corporation. All rights reserved. All Inprise and Borland brands and product names are trademarks or registered trademarks of Inprise Corporation. Other product names are trademarks or registered trademarks of their respective holders.

Printed in the U.S.A.

JDataStoreComponentLibraryReference3-s 2E2R0300

0001020304-9 8 7 6 5 4 3 2 1

PDF

Contents

Chapter 1 Introduction

1-1

Additional information	1-2
How this book is organized.	1-2
Documentation conventions	1-3
Contacting Borland developer support	1-3
Y2K.	1-3

Chapter 2 datastore package

2-1

Classes and components	2-2
Overview of classes in the <i>com.borland.datastore</i> package	2-2
DataStore component	2-3
DataStore variables.	2-8
DataStore constructors.	2-13
DataStore properties	2-13
DataStore methods	2-17
DataStore event listeners	2-19
DataStoreConnection component	2-19
DataStoreConnection variables	2-20
DataStoreConnection constructors	2-21
DataStoreConnection properties	2-21
DataStoreConnection methods	2-24
DataStoreException class	2-32
DataStoreException variables	2-32
DataStoreException properties	2-43
DataStoreException methods	2-43
FileOutputStream class	2-45
FileOutputStream properties	2-45
FileOutputStream methods	2-45
FileStream class	2-46
FileStream properties	2-47
FileStream methods	2-47
StreamProperties class.	2-48
StreamProperties properties.	2-48
StreamProperties methods	2-49
StreamVerifier class	2-50
StreamVerifier variables	2-50
StreamVerifier properties	2-52
StreamVerifier methods	2-52
TxException class	2-54
TxException variables	2-54
TxException properties	2-61
TxException methods	2-61

TxManager component.	2-63
TxManager variables	2-65
TxManager constructors	2-67
TxManager properties	2-67
TxManager methods	2-69

Chapter 3 datastore.jdbc package

3-1

Interfaces.	3-1
Classes and components	3-1
Overview of classes in the <i>com.borland.datastore.jdbc</i> package.	3-2
DataStoreDriver component.	3-2
DataStoreDriver constructors	3-4
DataStoreDriver properties.	3-4
DataStoreDriver methods	3-4
DataStoreServer component.	3-5
DataStoreServer variables	3-5
DataStoreServer constructors	3-6
DataStoreServer properties.	3-6
DataStoreServer methods	3-7
ServerConnection class.	3-9
ServerConnection variables	3-9
ServerConnection properties.	3-9
ServerConnection methods.	3-10
ServerStatus interface.	3-11
ServerStatus variables.	3-11
ServerStatusEvent class	3-12
ServerStatusEvent variables	3-12
ServerStatusEvent properties	3-12
ServerStatusEvent methods	3-13
ServerStatusListener interface.	3-13
ServerStatusListener methods	3-14

Chapter 4 dx.dataset package

4-1

Interfaces.	4-2
Classes and components	4-3
Overview of classes in the <i>com.borland.dx.dataset</i> package.	4-4
Aggregate operator classes.	4-4
Column-related classes	4-4
DataSet classes.	4-5
Descriptor classes	4-6
Event, listener, and adapter classes	4-6
Events for component writers	4-7

Exception classes	4-8	ColumnPaintAdapter class	4-59
Import/export classes	4-8	ColumnPaintAdapter properties	4-59
Provider and resolver classes	4-8	ColumnPaintAdapter methods	4-60
Row-related classes	4-9	ColumnPaintListener interface	4-60
Miscellaneous dataset classes	4-9	ColumnPaintListener methods	4-60
AccessEvent class	4-10	ColumnVariant class	4-61
AccessEvent variables	4-10	ColumnVariant variables	4-61
AccessEvent constructors	4-12	ColumnVariant constructors	4-62
AccessEvent properties	4-15	ColumnVariant properties	4-63
AccessEvent methods	4-16	ColumnVariant methods	4-64
AccessListener interface	4-17	CountAggOperator class	4-65
AccessListener methods	4-17	CountAggOperator variables	4-65
AggDescriptor class	4-17	CountAggOperator properties	4-65
AggDescriptor constructors	4-18	CountAggOperator methods	4-66
AggDescriptor properties	4-18	CustomAggOperator class	4-67
AggDescriptor methods	4-19	CustomAggOperator variables	4-68
AggOperator class (abstract)	4-20	CustomAggOperator properties	4-68
AggOperator variables	4-20	CustomAggOperator methods	4-68
AggOperator properties	4-21	CustomPaintSite interface	4-69
AggOperator methods	4-21	CustomPaintSite properties	4-70
CalcAggFieldsAdapter class	4-23	CustomPaintSite methods	4-71
CalcAggFieldsAdapter properties	4-24	DataChangeAdapter class	4-72
CalcAggFieldsAdapter methods	4-24	DataChangeAdapter properties	4-72
CalcAggFieldsListener interface	4-24	DataChangeAdapter methods	4-72
CalcAggFieldsListener methods	4-25	DataChangeEvent class	4-72
CalcFieldsListener interface	4-26	DataChangeEvent variables	4-73
CalcFieldsListener methods	4-26	DataChangeEvent constructors	4-74
CalcType interface	4-27	DataChangeEvent properties	4-74
CalcType variables	4-28	DataChangeEvent methods	4-75
CoerceFromListener interface	4-28	DataChangeListener interface	4-76
CoerceFromListener methods	4-29	DataChangeListener methods	4-76
CoerceToListener interface	4-30	DataFile class (abstract)	4-77
CoerceToListener methods	4-30	DataFile properties	4-77
Column component	4-31	DataFile methods	4-77
Column constructors	4-34	DataFileFormat class	4-78
Column properties	4-34	DataFileFormat variables	4-79
Column methods	4-51	DataFileFormat properties	4-79
Column event listeners	4-53	DataFileFormat methods	4-79
ColumnAware interface	4-53	DataModule interface	4-80
ColumnAware properties	4-54	DataRow class	4-80
ColumnChangeAdapter class	4-54	DataRow constructors	4-81
ColumnChangeAdapter properties	4-54	DataRow properties	4-82
ColumnChangeAdapter methods	4-54	DataRow methods	4-82
ColumnChangeListener interface	4-55	DataSet class (abstract)	4-86
ColumnChangeListener methods	4-55	DataSet properties	4-87
ColumnList component	4-56	DataSet methods	4-92
ColumnList constructors	4-57	DataSet event listeners	4-113
ColumnList properties	4-57	DataSetAware interface	4-114
ColumnList methods	4-58	DataSetAware properties	4-114

DataSetData class	4-114	MetaDataUpdate interface	4-177
DataSetData properties	4-115	MetaDataUpdate variables	4-178
DataSetData methods	4-115	MinAggOperator class	4-179
DataSetException class	4-116	MinAggOperator variables	4-179
DataSetException variables	4-117	MinAggOperator properties	4-180
DataSetException constructors	4-133	MinAggOperator methods	4-180
DataSetException properties	4-134	NavigationEvent class	4-181
DataSetException methods	4-134	NavigationEvent variables	4-181
DataSetView component	4-143	NavigationEvent constructors	4-181
DataSetView constructors	4-144	NavigationEvent properties	4-182
DataSetView properties	4-144	NavigationEvent methods	4-182
DataSetView methods	4-146	NavigationListener interface	4-183
DataSetView event listeners	4-151	NavigationListener methods	4-183
DxDispatch interface	4-151	OpenAdapter class	4-183
DxDispatch methods	4-152	OpenAdapter properties	4-183
EditAdapter class	4-152	OpenAdapter methods	4-184
EditAdapter properties	4-152	OpenListener interface	4-184
EditAdapter methods	4-152	OpenListener methods	4-184
EditListener interface	4-153	ParameterRow component	4-185
EditListener methods	4-154	ParameterRow constructors	4-186
ExceptionEvent class	4-160	ParameterRow properties	4-186
ExceptionEvent variables	4-160	ParameterRow methods	4-187
ExceptionEvent constructors	4-160	ParameterType class	4-191
ExceptionEvent properties	4-160	ParameterType variables	4-191
ExceptionEvent methods	4-161	ParameterType properties	4-192
ExceptionListener interface	4-162	ParameterType methods	4-192
ExceptionListener methods	4-162	PickListDescriptor class	4-193
LoadCancel interface	4-162	PickListDescriptor constructors	4-195
LoadCancel methods	4-163	PickListDescriptor properties	4-196
LoadEvent class	4-163	PickListDescriptor methods	4-197
LoadEvent variables	4-163	Provider class (abstract)	4-198
LoadEvent constructors	4-164	Provider properties	4-198
LoadEvent properties	4-164	Provider methods	4-199
LoadEvent methods	4-164	ProviderHelp class	4-201
LoadListener interface	4-165	ProviderHelp properties	4-201
LoadListener methods	4-165	ProviderHelp methods	4-201
LoadRowListener interface	4-165	ReadRow class (abstract)	4-205
LoadRowListener methods	4-166	ReadRow properties	4-205
Locate interface	4-166	ReadRow methods	4-206
Locate variables	4-166	ReadWriteRow class (abstract)	4-218
MasterLinkDescriptor class	4-168	ReadWriteRow properties	4-218
MasterLinkDescriptor constructors	4-171	ReadWriteRow methods	4-219
MasterLinkDescriptor properties	4-173	Resolver class (abstract)	4-232
MasterLinkDescriptor methods	4-175	Resolver properties	4-232
MaxAggOperator class	4-176	Resolver methods	4-232
MaxAggOperator variables	4-176	ResolverAdapter class	4-233
MaxAggOperator properties	4-176	ResolverAdapter properties	4-234
MaxAggOperator methods	4-176	ResolverAdapter methods	4-234

ResolverListener interface	4-235
ResolverListener methods	4-235
ResolverResponse component	4-239
ResolverResponse variables	4-240
ResolverResponse constructors	4-240
ResolverResponse properties	4-240
ResolverResponse methods	4-241
ResponseAdapter class	4-241
ResponseAdapter properties	4-242
ResponseAdapter methods	4-242
ResponseEvent class	4-242
ResponseEvent variables	4-242
ResponseEvent constructors	4-246
ResponseEvent properties	4-246
ResponseEvent methods	4-248
ResponseListener interface	4-249
ResponseListener methods	4-249
RowFilterListener interface	4-249
RowFilterListener methods	4-250
RowFilterResponse class	4-250
RowFilterResponse properties	4-251
RowFilterResponse methods	4-251
RowIterator class	4-252
RowIterator properties	4-253
RowIterator methods	4-255
RowStatus interface	4-261
RowStatus variables	4-261
Sort interface	4-262
Sort variables	4-262
SortDescriptor class	4-263
SortDescriptor constructors	4-264
SortDescriptor properties	4-266
SortDescriptor methods	4-268
StatusEvent class	4-270
StatusEvent variables	4-270
StatusEvent constructors	4-273
StatusEvent properties	4-273
StatusEvent methods	4-274
StatusListener interface	4-274
StatusListener methods	4-275
StorageDataSet component	4-275
StorageDataSet constructors	4-276
StorageDataSet properties	4-277
StorageDataSet methods	4-284
StorageDataSet event listeners	4-300
StoreClassFactory interface	4-302
StoreClassFactory methods	4-302
SumAggOperator class	4-303
SumAggOperator variables	4-303

SumAggOperator properties	4-303
SumAggOperator methods	4-304
TableDataSet component	4-305
TableDataSet constructors	4-306
TableDataSet properties	4-306
TableDataSet methods	4-307
TableDataSet event listeners	4-314
TextDataFile component	4-315
TextDataFile constructors	4-316
TextDataFile properties	4-316
TextDataFile methods	4-318
UpdateMode interface	4-319
UpdateMode variables	4-319
ValidationException class	4-320
ValidationException variables	4-320
ValidationException constructors	4-330
ValidationException properties	4-330
ValidationException methods	4-331
Variant component	4-333
Variant variables	4-334
Variant constructors	4-340
Variant properties	4-340
Variant methods	4-348
VariantException class	4-352
VariantException constructors	4-352
VariantException properties	4-352
VariantException methods	4-353

Chapter 5

dx.sql.dataset package	5-1
Interfaces	5-2
Classes and components	5-2
Overview of classes in the	
com.borland.dx.sql.dataset package	5-3
Connecting and data set classes	5-3
Event, listener, and adapter classes	5-3
Descriptor classes	5-4
Exception classes	5-4
Provider classes	5-4
Resolver classes	5-4
Miscellaneous dataset classes	5-5
ConnectionDescriptor class	5-5
ConnectionDescriptor constructors	5-5
ConnectionDescriptor properties	5-7
ConnectionDescriptor methods	5-9
ConnectionUpdateAdapter class	5-9
ConnectionUpdateAdapter properties	5-10
ConnectionUpdateAdapter methods	5-10

ConnectionUpdateEvent class	5-10	QueryResolver component	5-81
ConnectionUpdateEvent variables	5-11	QueryResolver constructors	5-82
ConnectionUpdateEvent constructors	5-11	QueryResolver properties	5-82
ConnectionUpdateEvent properties	5-12	QueryResolver methods	5-83
ConnectionUpdateEvent methods	5-12	QueryResolver event listeners	5-85
ConnectionUpdateListener interface	5-12	ResolutionException class	5-85
ConnectionUpdateListener methods	5-13	ResolutionException variables	5-86
Database component	5-13	ResolutionException constructors	5-89
Database variables	5-15	ResolutionException properties	5-90
Database constructors	5-16	ResolutionException methods	5-91
Database properties	5-16	ResolveError class	5-93
Database methods	5-21	ResolveError variables	5-94
Database event listeners	5-27	ResolveError properties	5-95
DefaultResolver interface	5-28	ResolveError methods	5-96
DefaultResolver methods	5-28	SQLDialect interface	5-96
Load interface	5-28	SQLDialect variables	5-96
Load variables	5-29	SQLResolutionManager component	5-97
OracleProcedureProvider class	5-30	SQLResolutionManager constructors	5-98
OracleProcedureProvider properties	5-31	SQLResolutionManager properties	5-98
OracleProcedureProvider methods	5-31	SQLResolutionManager methods	5-98
ProcedureDataSet class	5-32	SQLResolver class (abstract)	5-100
ProcedureDataSet properties	5-34	SQLResolver properties	5-100
ProcedureDataSet methods	5-37	SQLResolver methods	5-101
ProcedureDataSet event listeners	5-44	SQLResolver event listeners	5-102
ProcedureDescriptor class	5-45		
ProcedureDescriptor constructors	5-46	Chapter 6	
ProcedureDescriptor properties	5-48	dx.text package	6-1
ProcedureDescriptor methods	5-48	Interfaces	6-1
ProcedureProvider class	5-48	Classes and components	6-2
ProcedureProvider properties	5-49	Overview of classes in the	
ProcedureProvider methods	5-49	com.borland.dx.text package	6-2
ProcedureResolver component	5-52	ItemEditor classes	6-2
ProcedureResolver constructors	5-52	ItemFormatter classes	6-3
ProcedureResolver properties	5-52	Exception-related classes	6-4
ProcedureResolver methods	5-54	Placement-related classes	6-4
ProcedureResolver event listeners	5-55	Alignment class	6-4
QueryDataSet class	5-56	Alignment variables	6-5
QueryDataSet properties	5-59	Alignment properties	6-7
QueryDataSet methods	5-62	Alignment methods	6-7
QueryDataSet event listeners	5-70	BigDecimalFormatter class	6-7
QueryDescriptor class	5-71	BigDecimalFormatter constructors	6-8
QueryDescriptor constructors	5-73	BigDecimalFormatter properties	6-8
QueryDescriptor properties	5-75	BigDecimalFormatter methods	6-8
QueryDescriptor methods	5-76	BinaryFormatter component	6-10
QueryProvider component	5-77	BinaryFormatter constructors	6-10
QueryProvider constructors	5-77	BinaryFormatter properties	6-10
QueryProvider properties	5-77	BinaryFormatter methods	6-11
QueryProvider methods	5-78	BooleanFormat component	6-14
		BooleanFormat constructors	6-15

BooleanFormat properties	6-16	LongFormatter methods	6-51
BooleanFormat methods	6-16	ObjectFormatter component	6-53
BooleanFormatter component	6-19	ObjectFormatter constructors	6-53
BooleanFormatter constructors	6-19	ObjectFormatter properties	6-53
BooleanFormatter properties	6-19	ObjectFormatter methods	6-53
BooleanFormatter methods	6-19	ShortFormatter class	6-55
ByteFormatter class	6-21	ShortFormatter variables	6-55
ByteFormatter variables	6-21	ShortFormatter constructors	6-56
ByteFormatter constructors	6-21	ShortFormatter properties	6-56
ByteFormatter properties	6-21	ShortFormatter methods	6-56
ByteFormatter methods	6-21	SimpleFormatter component	6-58
DateFormatter component	6-23	SimpleFormatter constructors	6-58
DateFormatter constructors	6-23	SimpleFormatter properties	6-58
DateFormatter properties	6-23	SimpleFormatter methods	6-59
DateFormatter methods	6-24	StringFormatter component	6-62
DoubleFormatter class	6-25	StringFormatter constructors	6-62
DoubleFormatter constructors	6-25	StringFormatter properties	6-62
DoubleFormatter properties	6-26	StringFormatter methods	6-62
DoubleFormatter methods	6-26	TextFormat component	6-64
IntegerFormatter class	6-27	TextFormat variables	6-66
IntegerFormatter variables	6-28	TextFormat constructors	6-66
IntegerFormatter constructors	6-28	TextFormat properties	6-66
IntegerFormatter properties	6-28	TextFormat methods	6-67
IntegerFormatter methods	6-29	TimeFormatter component	6-69
InvalidFormatException class	6-30	TimeFormatter constructors	6-69
InvalidFormatException constructors	6-31	TimeFormatter properties	6-69
InvalidFormatException properties	6-31	TimeFormatter methods	6-70
InvalidFormatException methods	6-32	TimestampFormatter component	6-71
ItemEditMask interface	6-32	TimestampFormatter constructors	6-72
ItemEditMask methods	6-33	TimestampFormatter properties	6-72
ItemEditMaskState component	6-36	TimestampFormatter methods	6-72
ItemEditMaskState variables	6-36	VariantFormatStr class	6-74
ItemEditMaskState constructors	6-36	VariantFormatStr constructors	6-74
ItemEditMaskState properties	6-37	VariantFormatStr properties	6-76
ItemEditMaskState methods	6-37	VariantFormatStr methods	6-77
ItemEditMaskStr class	6-37	VariantFormatter class (abstract)	6-80
ItemEditMaskStr constructors	6-38	VariantFormatter properties	6-80
ItemEditMaskStr properties	6-39	VariantFormatter methods	6-81
ItemEditMaskStr methods	6-39		
ItemFormatStr class	6-40		
ItemFormatStr constructors	6-44		
ItemFormatStr properties	6-44		
ItemFormatStr methods	6-45		
ItemFormatter class (abstract)	6-47		
ItemFormatter properties	6-48		
ItemFormatter methods	6-49		
LongFormatter component	6-50		
LongFormatter constructors	6-50		
LongFormatter properties	6-51		

Chapter 7	
jb.io package	7-1
Interfaces	7-1
Classes and components	7-1
Overview of classes in the	
com.borland.jb.io package	7-2
AsciiInputStream class	7-3
AsciiInputStream constructors	7-3
AsciiInputStream properties	7-3
AsciiInputStream methods	7-4

AsciiOutputStream class	7-5	ArrayResourceBundle class (abstract)	8-3
AsciiOutputStream constructors	7-5	ArrayResourceBundle variables	8-4
AsciiOutputStream properties	7-5	ArrayResourceBundle properties	8-4
AsciiOutputStream methods	7-6	ArrayResourceBundle methods	8-5
EncodedInputStream class	7-7	BasicBeanInfo class (abstract)	8-6
EncodedInputStream constructors	7-7	BasicBeanInfo variables.	8-7
EncodedInputStream properties	7-7	BasicBeanInfo properties	8-11
EncodedInputStream methods	7-8	BasicBeanInfo methods	8-12
EncodedOutputStream class	7-9	ChainedException interface	8-13
EncodedOutputStream constructors	7-9	ChainedException properties	8-14
EncodedOutputStream properties	7-10	Diagnostic class	8-14
EncodedOutputStream methods	7-10	Diagnostic variables.	8-14
FastBufferedInputStream class	7-11	Diagnostic properties	8-15
FastBufferedInputStream variables.	7-11	Diagnostic methods	8-15
FastBufferedInputStream constructors.	7-11	DispatchableEvent class (abstract)	8-22
FastBufferedInputStream properties	7-12	DispatchableEvent variables	8-22
FastBufferedInputStream methods.	7-12	DispatchableEvent constructors	8-22
FastBufferedOutputStream class	7-15	DispatchableEvent properties	8-23
FastBufferedOutputStream variables	7-15	DispatchableEvent methods	8-23
FastBufferedOutputStream constructors.	7-15	ErrorResponse component.	8-24
FastBufferedOutputStream properties.	7-16	ErrorResponse variables	8-24
FastBufferedOutputStream methods.	7-16	ErrorResponse constructors	8-25
InputStreamToByteArray class	7-17	ErrorResponse properties	8-25
InputStreamToByteArray variables	7-17	ErrorResponse methods	8-26
InputStreamToByteArray constructors.	7-18	EventMulticaster class	8-27
InputStreamToByteArray properties	7-18	EventMulticaster variables	8-27
InputStreamToByteArray methods.	7-18	EventMulticaster properties	8-28
SimpleCharInputStream class (abstract)	7-19	EventMulticaster methods	8-28
SimpleCharInputStream properties	7-19	ExceptionChain component	8-30
SimpleCharInputStream methods	7-20	ExceptionChain constructors.	8-31
SimpleCharOutputStream class (abstract)	7-21	ExceptionChain properties	8-31
SimpleCharOutputStream properties	7-21	ExceptionChain methods.	8-31
SimpleCharOutputStream methods	7-21	ExceptionDispatch interface	8-32
		ExceptionDispatch methods	8-33
Chapter 8		ExceptionHandler interface	8-33
jb.util package	8-1	ExceptionHandler methods	8-33
Interfaces	8-1	FastStringBuffer component.	8-33
Classes and components	8-1	FastStringBuffer variables	8-34
Overview of classes in the		FastStringBuffer constructors	8-34
com.borland.jb.util package	8-2	FastStringBuffer properties.	8-35
Diagnostic classes	8-2	FastStringBuffer methods	8-36
Event-dispatching classes	8-2	Hex class	8-47
Exception-related classes	8-2	Hex variables	8-48
I/O classes	8-2	Hex properties	8-48
Internationalization classes	8-3	Hex methods.	8-48
Multicaster classes	8-3	LocaleUtil class	8-49
TriState interfaces.	8-3	LocaleUtil properties	8-49
Miscellaneous util classes	8-3	LocaleUtil methods	8-49

SearchPath class	8-50	Data type conversions during data providing.	A-3
SearchPath constructors	8-50	Mapping of JDBC data types to Variant data	
SearchPath properties	8-50	types	A-4
SearchPath methods	8-50	Data type coercions	A-4
TriStateProperty interface.	8-51		
TriStateProperty variables.	8-51		
VetoableDispatch interface	8-52		
VetoableDispatch methods	8-52		
VetoException component	8-53		
VetoException constructors	8-53		
VetoException properties	8-53		
VetoException methods	8-54		

Appendix A

Data type conversions A-1

Data type conversions during resolving.	A-1
Conversion from the user-specified data type to the default Variant data type	A-1
Mapping of Variant data types to JDBC data types	A-2
Data type coercions	A-3

Appendix B

String-based patterns (masks) B-1

Numeric data patterns	B-1
Date/time data patterns	B-3
Text data patterns	B-4

Index I-1

Introduction

The *JDataStore Component Library Reference* is a reference guide to the following packages, which provide a full-featured API for Java database application development:

datastore package	The <i>datastore</i> package provides features for persistent row storage and caching for JBuilder DataSet objects, Java Objects, and arbitrary files.
datastore.jdbc package	The <i>datastore.jdbc</i> package contains components for JDBC-specific database connectivity.
dbswing package	The <i>dbswing</i> package contains data-aware extensions to components in the <code>com.sun.java.swing</code> package. It also provides new components and an enriched feature set.
dx.dataset package	The <i>dx.dataset</i> package contains components for general data connectivity. It provides support for data-aware controls and visual development of data applications.
dx.sql.dataset package	The <i>dx.sql.dataset</i> package contains components for SQL-specific database connectivity.
dx.text package	The <i>dx.text</i> package contains classes for formatting data values and handling input validation.

This book is alphabetically organized by package. For each package, the classes, components, and interfaces are listed alphabetically. Classes intended for internal use and *BeanInfo* classes are not included in this book; however, they are listed at the beginning of each chapter.

Additional information

For additional information on these libraries, visit the JBuilder newsgroups at <http://www.borland.com/newsgroups>. Newsgroups are actively monitored by our support engineers as well as the JBuilder Research and Development team.

Additional Java-related information is available from various sources including:

- The Developer Support JBuilder FAQ (Frequently Asked Questions) page at <http://www.borland.com/devsupport/jbuilder/faq/>
- The Java FAQ page at <http://www.afu.com/javafaq.html>

How this book is organized

The chapters in this book are organized to give you quick access to the information you need in order to use the classes, components, and interfaces defined in the libraries.

The first section in each chapter contains an overview of the package and lists the classes, components and interfaces it contains. The classes, components, and interfaces in that package follow in alphabetical order. The documentation for a class, component, or interface contains the following sections:

- **Inheritance tree**, including one level of subclasses.
- **Description** of this class, component, or interface.
- **Variables** (fields) implemented in or inherited by this class, component, or interface.
- **Constructors** implemented in this class, component, or interface.
- **Properties** implemented in or inherited by in this class, component, or interface. In the Properties tables, one asterisk (*) indicates a read-only property; two asterisks (**) indicate a write-only property.
- **Methods** implemented in or inherited by this class, component, or interface.
- **Event listeners** implemented in this class, component, or interface.

The tables in each section list not only the variables, properties, and methods that are defined within that class, component, or interface, but also list all those that are inherited. If a table lists an entry as being in “this class,” but you cannot find the entry’s description in the section following the table, it is inherited. Using these tables, you can easily see what methods and fields are available, not just those implemented within that class, component, or interface.

Documentation conventions

The documentation for JBuilder uses the typefaces and symbols described in the table below to indicate special text.

Typeface	Meaning
Monospace type	Monospaced type represents text as it appears on screen or in Java code. It also represents anything you must type.
[]	Square brackets in text or syntax listings enclose optional items. Do not type the brackets.
< >	Angle brackets in text or syntax listings indicate a variable string; type in a string appropriate for your code. Do not type the angle brackets. Angle brackets are also used for HTML tags.
Boldface	Boldfaced words in text represent Java reserved words.
<i>Italics</i>	Italicized words represent Java identifiers, such as names of variables, classes, interfaces, components, properties, methods, and events. Italics are also used for new terms being defined.
<i>Keycaps</i>	This typeface indicates a key on your keyboard. For example, “Press <i>Esc</i> to exit a menu.”

Contacting Borland developer support

Borland offers a variety of support options. These include free services on the Internet, where you can search our extensive information base and connect with other users of Borland products. In addition, you can choose from several categories of support, ranging from support on installation of the Borland product to fee-based consultant-level support and detailed assistance.

For more information about Borland’s developer support services, please see our Web site at <http://www.borland.com/devsupport>, call Borland Assist at (800) 523-7070, or contact our Sales Department at (831) 431-1064.

When contacting support, be prepared to provide complete information about your environment, the version of the product you are using, and a detailed description of the problem.

Y2K

For information about year 2000 issues and our products, see the following URL: <http://www.borland.com/about/y2000/>.

datastore package

The *datastore* package collects the functionality for the DataStore, a high-performance, small-footprint, 100% Pure Java embeddable database; that also provides persistent row storage and caching for JBuilder *DataSet* objects, Java Objects, and arbitrary files. Each DataStore is a single file (within the host platform's file system) with its own internal hierarchical directory.

DataStore provides solutions for a number of data storage problems. For example, you can:

- Embed SQL-92-compliant database functionality directly into your application, without having to rely on an external database engine. Databases can be accessed through the DataStore JDBC driver, or through the DataExpress API. DataStore supports most JDBC data types, including Java Object. Transactional multi-user access and crash recovery is available.
- Serialize all your application's objects and file streams into a single physical file for convenience and portability.
- Enable mobile and off-line applications. Using DataExpress datasets, DataStore asynchronously replicates and caches data from an arbitrary data source (for example, a database server, a CORBA application server, SAP, BAAN, etc.), allows access and updates, and resolves changes back into the data source.
- Increase the performance of on-line applications with large datasets by using a *DataStore* instead of the default *MemoryStore*.

The following classes and components in this package are intended for public use:

- DataStore
- DataStoreConnection
- DataStoreException
- FileOutputStream

- FileStream
- StreamProperties
- StreamVerifier
- TxException
- TxManager

The following *BeanInfo* classes are in this package:

- DataStoreBeanInfo
- DataStoreConnectionBeanInfo
- TxManagerBeanInfo

For more information, visit the database newsgroup. Details on newsgroups can be found at <http://www.borland.com/newsgroups>. The database newsgroup is dedicated to issues about writing database applications and is actively monitored by our support engineers as well as the Development team.

Classes and components

DataStore	DataStoreBeanInfo	DataStoreConnection
DataStoreConnectionBeanInfo	DataStoreException	FileOutputStream
FileStream	StreamProperties	StreamVerifier
TxException	TxManager	TxManagerBeanInfo

Overview of classes in the *com.borland.datastore* package

DataStore	Represents a DataStore file; used when creating a new DataStore. Extends <i>DataStoreConnection</i> .
DataStoreConnection	A single transactional context for a <i>DataStore</i> . Each transaction has its own <i>DataStoreConnection</i> instance.
DataStoreException	Extends the <i>DataSetException</i> class with error and other notification messages that are specific to a <i>DataStore</i> .
FileOutputStream	Provides read, write, and seek access for file streams stored in a <i>DataStore</i> .
FileStream	Provides read, write, and seek access for file streams stored in a <i>DataStore</i> .
StreamProperties	Collects properties for a <i>DataStore</i> stream.
StreamVerifier	Used to verify the integrity of a file or dataset stream.

TxException	Extends the <i>DataSetException</i> class with error and other notification messages that are specific to transactional access to a DataStore.
TxManager	The DataStore transaction manager; manages transactional access for a single DataStore.

DataStore component

datastore package

Extends	com.borland.datastore.DataStoreConnection
Implements	com.borland.dx.dataset.Designable, com.borland.dx.dataset.Store

The *DataStore* component represents a physical DataStore file. See the package documentation for a brief overview of DataStore features, and the *JDataStore Programmer's Guide* for detailed usage information.

The functionality of previous versions of the *DataStore* has been divided into this class and its superclass, the *DataStoreConnection* class, which represents connections to a DataStore file. Multiple connections from the same process are now supported. Multi-user access to table streams in the DataStore is supported through the remote JDBC driver; users of the remote driver communicate with a single DataStore JDBC server process.

Using *DataStore* instead of *MemoryStore*

By default, once data is loaded into a JDataStore component the storage mechanism is in-memory storage through the use of a *MemoryStore*. Alternate storage systems such as the *DataStore* are allowed by setting the *StorageDataSet* object's *store* property. (Currently *MemoryStore* and *DataStore* are the only implementations of the *Store* interface required by the *store* property.)

The main advantage of *DataStore* over *MemoryStore* is persistence, which enables off-line computing. In addition, you can increase the performance of any application with large *StorageDataSets*. *StorageDataSets* using *MemoryStore* have a small performance edge over *DataStore* for small number of rows, however, *DataStore* stores data and indexes in an extremely compact format. As the number of rows in a *StorageDataSet* increases, using a *DataStore* is much more performant and requires much less memory than using a *MemoryStore*.

Whether the data's storage is *MemoryStore* or *DataStore* does not effect how you work with a *DataSet* or other data-aware controls connected to the *DataSet*. Persistence, however, does require use of Java serialization (java.io.Serializable). If this is not possible, you cannot use *DataStore* components and should use the default in-memory storage mechanism.

Creating the DataStore file

There are two ways to create a DataStore file, depending on whether you are using the DataStore with a *StorageDataSet* (either as persistent off-line storage or as an embedded database through the DataExpress API). Either way, the name of the DataStore file is controlled by the *DataStore* object's *fileName* property, and starts out with the same general steps:

- 1 Instantiate the *DataStore*.
- 2 Set the *fileName* for the DataStore file.
- 3 Set any other properties that must be set at create-time, like the *blockSize*, if desired.
- 4 To make the DataStore transactional:
 - 1 Instantiate a *TxManager*
 - 2 Set its properties if desired
 - 3 Assign it to the *DataStore*'s *txManager* property
 - 4 Assign a name to the *DataStore*'s *userName* property

When using the DataStore with a *StorageDataSet*, you continue with these two steps:

- 1 Assign the *DataStore* to the *StorageDataSet*'s *store* property.
- 2 Assign a name to the *StorageDataSet*'s *storeName* property. This name will identify the *StorageDataSet*'s data inside the DataStore.

This will create the DataStore when necessary (at design time, when using the visual design tools). The alternative is to create the DataStore manually, which is more likely when using the JDBC driver to access the DataStore. There is only one additional step:

- 1 Call the *DataStore*'s *create* method.

Because the *DataStore* is a subclass of *DataStoreConnection*, the *create* method (if successful) results in an open connection to the newly-created DataStore through the *DataStore* object. You can then use methods in the *datastore* package (the DataStore API) and a local JDBC driver simultaneously to access the contents of the DataStore.

Opening a DataStore file

The steps for opening an existing DataStore are very similar to the steps for creating one. For either type of use, it is not necessary to instantiate a *TxManager* to assign to the *txManager* property; this will be done automatically when you open a transactional DataStore. It doesn't hurt if you do assign one, and assigning one with different properties before you open the DataStore is how you change transactional properties.

Otherwise, the steps for using an existing DataStore with a *StorageDataSet* are identical to the steps when creating one. Also, you may use the same

DataStore with multiple *StorageDataSets*; each one must have a different (uniquely identifying) *storeName* property. This name is case-sensitive.

Opening a DataStore manually has two differences:

- You call the *open* method instead of the *create* method.
- Unless you want to perform maintenance or change one of the properties of the DataStore file, you should use the *DataStoreConnection* class to open a connection to an existing DataStore. This is the superclass of *DataStore* which actually contains the *fileName* property and *open* method.

If you're using the JDBC driver to access the DataStore, you don't need to open the DataStore manually.

DataStore directory DataSet

The directory structure of the *DataStore* associates a name and various directory status with a particular data stream. (The "/" character is the directory separator in the DataStore's internal file system.) *DataStore* currently has these public data stream categories:

- **Table:** Embedded database data and persistent cached data from any DataExpress *DataSet* that extends from *StorageDataSet* (for example, *QueryDataSet*, *ProcedureDataSet*, and *TableDataSet*). All *StorageDataSet* features are supported including master detail, picklists, constraints, calc fields, sorting, filtering, and aggregation.
- **File:** *DataStore* component provides methods to create and open arbitrary file streams (*createFileStream* and *openFileStream*). These methods return a *FileStream* that extends the Java *InputStream* class with additional *seek()* and *write()* methods.

Data of type Object can be stored in the *DataStore* as a file stream. The *DataStore* component provides convenience methods to serialize and deserialize them through the *writeObject()* and *readObject()* methods. If these Objects do not support Java serialization, an *Exception* is raised when the *DataStore* attempts to save the Object. Also, the class must exist on the classpath when the *DataStore* attempts to read an Object. (You can also persist Java Objects in a *DataStore* by using a *DataSet* with a *Column* that has an Object data type.)

You can access a *DataStore's* directory structure by calling its *openDirectory()* method. This returns a *TableDataSet* with several columns for the name, data stream category, modification time, size, delete status, and so on. The directory structure for the *DataStore's* file system typically contains multiple levels that you can traverse using this *TableDataSet*.

Call *deleteStream()* to delete a stream. The contents of these streams are reallocated as new storage is needed for existing streams. Although streams can be undeleted, the entire contents may not be present if space was already used for new allocations.

Time values stored in the *DataStore*'s directory *DataSet* and in Time, Date, and TimeStamp columns are recorded in UTC time.

See the *openDirectory()* method for more information on the directory *DataSet*.

Verifying, repairing, and upgrading a DataStore

Check the *DataStore*'s *consistent* property to see if the contents of the *DataStore* file may have been corrupted. One possible cause of corruption is calling *System.exit()* without properly closing the connections to the *DataStore*. Certain exceptions may also indicate possible or definite damage to the contents of a *DataStore*.

If the *DataStore* was transactional, and the log files have been lost or damaged, set the *readOnly* property to **true** before attempting to open the *DataStore*.

You may check the integrity of the *DataStore* with a *StreamVerifier* object. If it indicates there are errors, use the *DataStoreConnection.copyStreams* method to attempt to copy all the streams to a new *DataStore*. *copyStreams* tries to repair damaged streams and copy them correctly.

copyStreams is also the way to upgrade an older version of *DataStore* to the current version. Simply copy all the streams to a new *DataStore* file.

The following code fragment shows the basic steps for copying all the streams to a new *DataStore*:

```
// datastore is the source, which has already been opened
DataStore newStore;

// First create a copy with the same basic structure
newStore = new DataStore();
newStore.setFileName( "COPY_" + datastore.getFileName() );
newStore.setBlockSize( datastore.getBlockSize() );
newStore.create();

// Copy all streams, ignoring errors
datastore.copyStreams(
    "", // From root directory
    "**", // All streams
    newStore, // To new store
    "", // To root directory
    DataStore.COPY_IGNORE_ERRORS, // Ignore individual errors
    System.out ); // Status messages to console

// Be sure to close new DataStore
newStore.shutdown();
```

Data type coercions

When the data type of a *Column* component in a *StorageDataSet* that is bound to a *DataStore* is changed, type coercions occur when going from one type to another. Note that this form of coercion is different than the ones performed when providing and resolving data (see *CoerceToListener* and *CoerceFromListener*).

The following table describes what happens when a data type is coerced to another data type. The data types on the left indicate the original data type of the *Column* with the data types listed along the top of the table indicating the new data type of the *Column*.

Table 2.1 DataStore data type coercions

From \ To	Big Decimal	Double	Float	Long	int	Short	boolean	Time	Date	Time stamp	String	Input Stream	Object
BigDecimal	None	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	OK	Loss	Loss
Double	Prec	None	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	OK	Loss	Loss
Float	Prec	OK	None	Prec	Prec	Prec	Prec	Prec	Prec	Prec	OK	Loss	Loss
Long	OK	Prec	Prec	None	Prec	Prec	Prec	Prec	Prec	Prec	OK	Loss	Loss
int	OK	OK	Prec	OK	None	Prec	Prec	Prec	Prec	Prec	OK	Loss	Loss
Short	OK	OK	OK	OK	OK	None	Prec	Prec	Prec	Prec	OK	Loss	Loss
boolean	OK	OK	OK	OK	OK	OK	None	Prec	Prec	Prec	OK	Loss	Loss
Time	Prec	Prec	Prec	Prec	Prec	Prec	Prec	None	Prec	Prec	OK	Loss	Loss
Date	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	None	Prec	OK	Loss	Loss
Timestamp	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	Prec	None	OK	Loss	Loss

Where the table values represent the following:

- Loss - All data lost in this coercion.
- None - No coercion necessary.
- Prec - Potential precision loss with this coercion.
- OK - No data lost with this coercion.

DataStore specifications

The specifications of a *DataStore* are as follows:

Capacity	Max DataStore file size: Blocksize * 2 gigabytes (approximately 4,400 gigabytes)
Default block size	4k
Maximum rows	4 billion rows per table
Maximum row length	1/3 block size Note: long values are stored as Strings and Binary Large Objects (BLOBs). BLOBs are limited to 2 gigabytes each.

DataStore variables

Variable	Defined in
ACTIVE_STATE	this class
AGG_STREAM	this class
COPY_CASE_SENSITIVE	com.borland.datastore.DataStoreConnection
COPY_IGNORE_ERRORS	com.borland.datastore.DataStoreConnection
COPY_OVERWRITE	com.borland.datastore.DataStoreConnection
DELETED_STATE	this class
DELETED_STREAM	this class
DIR_ACCESS	this class
DIR_BLOB_LENGTH	this class
DIR_DEL_TIME	this class
DIR_ID	this class
DIR_LENGTH	this class
DIR_MOD_TIME	this class
DIR_PROPERTIES	this class
DIR_STATE	this class
DIR_STORE_NAME	this class
DIR_TYPE	this class
FETCH_STREAM	this class
FILE_STREAM	this class
HIDDEN_STREAM	this class
INSERTED_STREAM	this class
MAX_BIGDECIMAL_SCALE	this class
ORIGINALS_STREAM	this class
SECOND_INDEX_STREAM	this class
STORE_VERSION	this class
TABLE_FILE_STREAM	this class
TABLE_STREAM	this class
VERSION_3_0	this class
VERSION_4_0	this class

ACTIVE_STATE

```
public static final short ACTIVE_STATE = 0x1
```

Constant that designates a permanent, active stream. One of the possible values stored in the State column of the directory *DataSet* (as returned by the *openDirectory()* method).

AGG_STREAM

```
public static final short AGG_STREAM = (short)5|TABLE_STREAM|HIDDEN_STREAM
```

Constant that designates a hidden, internal stream used to maintain aggregates in a *TABLE_STREAM*. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

DELETED_STATE

```
public static final short DELETED_STATE = 0x4
```

Constant that designates a deleted stream. One of the possible values stored in the State column of the *DataStore's* directory *DataSet* (as returned by the *openDirectory()* method).

DELETED_STREAM

```
public static final short DELETED_STREAM = (short)2|TABLE_STREAM|HIDDEN_STREAM
```

Constant that designates a hidden, internal stream used to store *DataSet* (*TABLE_STREAM*) deleted rows. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

DIR_ACCESS

```
public static final String DIR_ACCESS = "Access"
```

Reserved for future use.

DIR_BLOB_LENGTH

```
public static final String DIR_BLOB_LENGTH = "BlobLength"
```

Constant that refers to the BlobLength Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores the length (in bytes) of the table stream's blobs.

DIR_DEL_TIME

```
public static final String DIR_DEL_TIME = "DelTime"
```

Constant that refers to the DeleteTime Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores the UTC time a stream was deleted.

DIR_ID

```
public static final String DIR_ID = "Id"
```

Constant that refers to the Id Column of the directory *DataSet* (returned by the *openDirectory()* method) that is a unique identifier for a stream.

DIR_LENGTH

```
public static final String DIR_LENGTH = "Length"
```

Constant that refers to the Length Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores the length in bytes of a stream.

DIR_MOD_TIME

```
public static final String DIR_MOD_TIME = "ModTime"
```

Constant that refers to the ModTime Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores the last UTC time a stream was modified.

DIR_PROPERTIES

```
public static final String DIR_PROPERTIES = "Properties"
```

Constant that refers to the Properties Column of the directory *DataSet* (returned by the *openDirectory()* method) that lists persisted properties and events for a *DataSet* stream.

DIR_STATE

```
public static final String DIR_STATE = "State"
```

Constant that refers to the State Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores one of the *DataStore.*_STATE* values.

DIR_STORE_NAME

```
public static final String DIR_STORE_NAME = "StoreName"
```

Constant that refers to the StoreName Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores the name for a stream. The name may be up to 192 bytes long.

DIR_TYPE

```
public static final String DIR_TYPE = "Type"
```

Constant that refers to the Type Column of the directory *DataSet* (returned by the *openDirectory()* method) that stores one of the DataStore.*_STREAM values.

FETCH_STREAM

```
public static final short FETCH_STREAM = (short)6|TABLE_STREAM|HIDDEN_STREAM
```

Constant used to designate a hidden, internal stream that tracks detail groups (keys) fetched for a detail *DataSet* (*TABLE_STREAM*). One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

FILE_STREAM

```
public static final short FILE_STREAM = 0x2000
```

Attribute for a file stream or serialized object stream. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

See also *createFileStream()*, *openFileStream()*, *writeObject()*, *readObject()* methods

HIDDEN_STREAM

```
public static final short HIDDEN_STREAM = 0x4000
```

Attribute that designates a stream for internal use. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

INSERTED_STREAM

```
public static final short INSERTED_STREAM = (short)3|TABLE_STREAM|HIDDEN_STREAM
```

Constant that refers to a hidden stream used to index *TABLE_STREAM* inserted rows. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

MAX_BIGDECIMAL_SCALE

```
public static final int MAX_BIGDECIMAL_SCALE = 127
```

Maximum scale that can be used for *BigDecimal* columns. Use of larger scales will be reduced to this scale.

ORIGINALS_STREAM

```
public static final short ORIGINALS_STREAM = (short)4|TABLE_STREAM|HIDDEN_STREAM
```

Constant that designates a hidden stream used to store *TABLE_STREAM* original rows and index updated rows. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

SECOND_INDEX_STREAM

```
public static final short SECOND_INDEX_STREAM = (short)7|TABLE_STREAM|HIDDEN_STREAM
```

Constant that designates a hidden stream which maintains a secondary index of a *TABLE_STREAM*, including sorted and/or filtered views of its associated *TABLE_STREAM*. There can be one or more *SECOND_INDEX_STREAMS*. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

STORE_VERSION

```
public static final int STORE_VERSION = 0x040000
```

Indicates what the version of a newly created DataSet file will be.

TABLE_FILE_STREAM

```
public static final short TABLE_FILE_STREAM = (short)1|TABLE_STREAM|HIDDEN_STREAM
```

Constant used as an attribute for a file stream or serialized object stream stored inside a *TABLE_STREAM*. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

See also *createFileStream()*, *openFileStream()*, *writeObject()*, *readObject()* methods

TABLE_STREAM

```
public static final short TABLE_STREAM = (short)0x8000
```

Constant used as an attribute for a table stream. *DataSets* are persisted with this attribute. One of the possible values stored in the Type column of the directory *DataSet* (as returned by the *openDirectory()* method).

VERSION_3_0

```
public static final int VERSION_3_0 = 0x3000F
```

Constant used for the 3.0 version number.

VERSION_4_0

```
public static final int VERSION_4_0 = STORE_VERSION
```

Constant used for the 4.0 version number.

DataStore constructors

DataStore()

```
public DataStore()
```

Constructs a *DataStore* component configured for a default block size of 4KB and a maximum sort buffer size of 12MB. Change the *blockSize* and *maxSortBuffer* properties if desired before creating a new DataStore file with *create()*.

DataStore properties

Property	Implemented in
blockSize	this class
class*	java.lang.Object
consistent*	this class
dataStore*	com.borland.datastore.DataStoreConnection
fileName	com.borland.datastore.DataStoreConnection
locale	this class
lockWaitTime	com.borland.datastore.DataStoreConnection
maxRowLength*	this class
maxRowLocks	this class
maxSortBuffer	this class
minCacheSize	this class
open*	com.borland.datastore.DataStoreConnection
openMode	this class
readOnly	this class
readOnlyTx	com.borland.datastore.DataStoreConnection
readOnlyTxDelay	com.borland.datastore.DataStoreConnection
saveInterval	this class
saveMode	this class
storeInternals*	com.borland.datastore.DataStoreConnection
tempDirName	this class
txIsolation	com.borland.datastore.DataStoreConnection
txManager	this class
userName	com.borland.datastore.DataStoreConnection
version*	com.borland.datastore.DataStoreConnection

blockSize

```
public final int getBlockSize()
public final synchronized void setBlockSize(int blockSize)
```

Specifies the block size of the *DataStore* file, expressed in kilobytes. Unless specifically set, this property defaults to 4k. This property has no effect on an existing *DataStore* and is used only when creating a new *DataStore*. On failure, the *setBlockSize* method throws a *DataSetException*.

consistent

```
public final boolean isConsistent()
```

Returns **false** if the *DataStore* was ever shutdown improperly. Improper shutdown (such as calling *System.exit()* without closing a *DataStore*) can cause a *DataStore* to become corrupted. If the *DataStore* is using a *TxManager*, the *TxManager* will bring the *DataStore* back to an action- and transaction-consistent state when the *DataStore* is reopened. *isConsistent* can also be set back to **true** if a *StreamVerifier* can successfully verify the entire *DataStore*. This leaves the *DataStore* action-consistent, but not transaction-consistent.

locale

```
public Locale getLocale()
public synchronized void setLocale(Locale locale)
```

Specifies the locale associated with this *DataStore*. *StorageDataSet* components use this property when the *locale* property is not set at the *StorageDataSet* level.

maxRowLength

```
public final int getMaxRowLength()
```

Read-only property that returns the maximum length in bytes for a row stored in this *DataStore*. Note that Columns with large storage requirements like long Strings, Objects or BLOBs are stored separately and are not limited by this length.

maxRowLocks

```
public final int getMaxRowLocks()
public final void setMaxRowLocks(int maxRowLocks)
```

Maximum number of row locks (used for repeatable read transactions) allowed for this *DataStore*.

maxSortBuffer

```
public final int getMaxSortBuffer()
public final void setMaxSortBuffer(int maxSortBuffer)
```

Determines the maximum memory in bytes that is used for the buffer when performing sorting operations such as building indexes. Increasing this number may improve the performance of very large sorts. The minimum value for this property is 32×1024 .

Note Sorts that cannot be done in memory use a disk-based merge sort.

minCacheSize

```
public final int getMinCacheSize()
public final void setMinCacheSize(int size)
```

Determines the number of cache entries in the cache. Note that all *DataStore* components share the same cache. Also, if many streams are held open, the *DataStore* will increase the cache beyond this setting as needed.

openMode

```
public final String getOpenMode()
public final synchronized void setOpenMode(String mode)
```

Controls whether streams are opened in read/write (“rw”) or read-only (“r”) mode. If the *readOnly* property is **true**, no changes will be allowed, superseding this property.

readOnly

```
public final boolean isReadOnly()
public final void setReadOnly(boolean readOnly)
```

If **true**, only read access is allowed for this *DataStore*.

Read-only mode can be used to access a transactional *DataStore* when the log files are lost. After setting *readOnly*, *open* the *DataStore* and use *copyStreams* to copy the data to a new *DataStore*. See the section on verifying and repairing *DataStores* in the About section for more details.

saveInterval

```
public final long getSaveInterval()
public final void setSaveInterval(long interval)
```

Determines the time interval for asynchronous cache block saving for this *DataStore* component. The interval value is expressed in milliseconds and must be greater than or equal to 100 and less than or equal to 2500. Both the setter and getter may throw a *DataSetException* when applicable.

saveMode

```
public final int getSaveMode()
public final void setSaveMode(int mode)
```

Determines when cached changes to the *DataStore* are saved to disk. This property is ignored for transactional *DataStores* (their *TxManager* provides crash recovery). Set this property to one of the following:

- 0 - only save cache blocks asynchronously with the daemon thread. This option provides the best performance, but cached blocks are not guaranteed to be saved until the *save()* or *shutdown()* methods are called on the *DataStore*.
- 1 - extends mode 0 by saving cached blocks when a *DataStore* block is allocated or deleted. This is the default setting, which provides more safety by saving cached blocks more frequently.
- 2 - extends mode 1 by saving cached blocks when any update is made to a *DataStore*. This setting provides the most safety by saving cache blocks whenever they are modified. Use this when debugging to ensure all modified cache blocks are constantly saved.

This property only affects *FileStream* classes when the *FileStream* is first created. *FileStream.write()* operations are currently only saved by the daemon thread or an explicit call to *DataStore.save()*. Note that this property can be set dynamically without closing the *DataStore*.

tempDirName

```
public final String getTempDirName()
public final void setTempDirName(String tempDirName)
```

Specifies the directory for temporary tables created by the query engine, and temporary files for operations such as large sorts. If this property is not set, the directory of the *fileName* property is used for temporary files.

txManager

```
public final TxManager getTxManager()
public final void setTxManager(TxManager tm)
```

The transaction manager for the *DataStore*. If this property is **null** when the *DataStore* is created, it will not be transactional. It can be made transactional anytime in the future by setting this property before opening the *DataStore*.

DataStore methods

Method	Implemented in
clone()	java.lang.Object
close()	com.borland.datastore.DataStoreConnection
closeDirectory()	com.borland.datastore.DataStoreConnection
closeStream(java.lang.String)	com.borland.datastore.DataStoreConnection
commit()	com.borland.datastore.DataStoreConnection
copyStreams(java.lang.String, java.lang.String, com.borland.datastore.DataStoreConnection, java.lang.String, int, java.io.PrintStream)	com.borland.datastore.DataStoreConnection
create()	this class
createFileStream(java.lang.String)	com.borland.datastore.DataStoreConnection
deleteStream(java.lang.String)	com.borland.datastore.DataStoreConnection
dropIndexStream(com.borland.dx.dataset.ReadRow)	com.borland.datastore.DataStoreConnection
dumpLocks(java.io.PrintStream)	com.borland.datastore.DataStoreConnection
dumpTxLog(java.io.PrintStream, long)	com.borland.datastore.DataStoreConnection
equals(java.lang.Object)	java.lang.Object
fileExists(java.lang.String)	com.borland.datastore.DataStoreConnection
fileLength()	this class
finalize()	this class
getProductVersion()	com.borland.datastore.DataStoreConnection
getStreamProperties (com.borland.dx.dataset.ReadRow)	com.borland.datastore.DataStoreConnection
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.datastore.DataStoreConnection
openDirectory()	com.borland.datastore.DataStoreConnection
openFileStream(java.lang.String)	com.borland.datastore.DataStoreConnection
readObject(java.lang.String)	com.borland.datastore.DataStoreConnection
renameStream(java.lang.String, java.lang.String)	com.borland.datastore.DataStoreConnection
rollback()	com.borland.datastore.DataStoreConnection
save()	this class
shutdown()	this class
sortTable(java.lang.String, java.lang.String, com.borland.dx.dataset.SortDescriptor)	com.borland.datastore.DataStoreConnection
sync()	this class
tableExists(java.lang.String)	com.borland.datastore.DataStoreConnection

Method	Implemented in
toString()	java.lang.Object
transactionStarted()	com.borland.datastore.DataStoreConnection
undeleteStream(com.borland.dx.dataset. ReadRow)	com.borland.datastore.DataStoreConnection
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
writeObject(java.lang.String, java.lang.Object)	com.borland.datastore.DataStoreConnection

create()

public final synchronized void create()

Creates a new *DataStore* file using the *fileName* property. If a *TxManager* has been assigned to the *txManager* property (which then requires that the *userName* property also be set), the *DataStore* will be transactional, and the corresponding log files will be created as well. A *DataSetException* is thrown if the file specified in the *fileName* property already exists.

See also *DataStoreConnection.open* method

fileLength()

public final synchronized long fileLength()

Returns the length of the *DataStore* file in bytes.

finalize()

public final void finalize()

Closes the *DataStore* if it is still open.

Overrides java.lang.Object.finalize()

save()

public final void save()

Saves the contents of the cache to the *DataStore* file. On error, this method throws a *DataSetException*.

shutdown()

public final void shutdown()

Flushes the data cache, closes all connections to the *DataStore*, and closes the *DataStore*. On failure, this method throws a *DataSetException*. If an *IOException* is generated, it is thrown via a chained *DataSetException*.

sync()

```
public void sync()
```

Forces DataStore data to be written to disk. For a non-transactional DataStore, this guarantees that all changes have been written safely.

DataStore event listeners

This component is a source for the following event sets.

response

```
public final void addResponseListener(ResponseListener listener)
public final void removeResponseListener(ResponseListener listener)
```

DataStoreConnection component

datastore package

Extends java.lang.Object

Extended by com.borland.datastore.DataStore

Implements com.borland.dx.dataset.Designable, com.borland.dx.dataset.Store

A *DataStoreConnection* is used to access a *DataStore*. Multiple *DataStoreConnection* objects can connect to the same *DataStore* from the same process. Each connection provides a separate transactional context.

Because the *DataStore* subclasses *DataStoreConnection*, each *DataStore* has its own built-in connection. If you only need a single connection, you can instantiate the *DataStore* object, and use the connection through the properties and methods inherited from *DataStoreConnection*. This technique is more likely if you are creating the *DataStore* file, or when you are setting the physical properties of the *DataStore* itself.

You do not need to instantiate a *DataStore* object to access a *DataStore* file; the *DataStoreConnection* will do this for you automatically, opening a connection to the *DataStore* file specified in the *fileName* property. *DataStoreConnection* implements the *Store* interface, so it can be used in the *store* property setting of a *StorageDataSet*.

Opening a Connection

The minimum steps for opening a connection to a DataStore are:

- 1 Instantiate a *DataStoreConnection* object.
- 2 Set the *fileName* property of the *DataStoreConnection*.
- 3 If the DataStore is transactional, you must also set the *userName* property.
- 4 Call its *open* method.

Managing Transactions

A transactional *DataStore* is one that was created with a *TxManager* (the transaction manager). Once a *DataStore* is created, it remembers whether it is transactional and where its log files are kept, so that when it is opened again later, a *TxManager* will be instantiated automatically if necessary.

The *rollback* method will undo all changes made in the current connection since the last *rollback()* or *commit()* and terminate the transaction.

Directory Entry Parameters

A number of methods use a *dirEntry* parameter to indicate which stream to operate upon. This parameter is of the type *com.borland.dx.dataset.ReadRow*, and is usually an instance of one of its subclasses:

- A *StorageDataSet* returned by the *openDirectory* method.
- A *DataSetView* of such a *StorageDataSet*.
- A *DataRow* constructed from a row in one of those other *DataSets*.

For a *DataSet* with more than one row, the action is performed upon the stream identified by the current row.

DataStoreConnection variables

Variable	Defined in
COPY_CASE_SENSITIVE	this class
COPY_IGNORE_ERRORS	this class
COPY_OVERWRITE	this class

COPY_CASE_SENSITIVE

public static final int COPY_CASE_SENSITIVE = 0x2

By default all match comparisons for the *sourcePattern* parameter to *copyStreams()* are case-insensitive. When this is used in the *option* parameter, matches use strict case-sensitivity.

COPY_IGNORE_ERRORS

public static final int COPY_IGNORE_ERRORS = 0x4

By default any errors that are encountered during *copyStreams* terminate the copy operation. If this option is set, the copy operation will skip the stream that cannot be copied and attempt to copy the remaining streams. The name of the stream that caused the error and an exception stack trace will be sent to the output status stream (if one is specified).

COPY_OVERWRITE

```
public static final int COPY_OVERWRITE = 0x1
```

When used in the *option* parameter of *copyStreams()*, existing streams will be overwritten. By default, a *DataStoreException* will be thrown if the target stream already exists.

DataStoreConnection constructors

DataStoreConnection()

```
public DataStoreConnection()
```

Constructs a *DataStoreConnection* component. To connect to a *DataStore*, set the *fileName* property and call the *open()* method.

DataStoreConnection properties

Property	Implemented in
class*	java.lang.Object
dataStore*	this class
fileName	this class
locale*	this class
lockWaitTime	this class
open*	this class
readOnlyTx	this class
readOnlyTxDelay	this class
storeInternals*	this class
txIsolation	this class
userName	this class
version*	this class

dataStore

```
public final DataStore getDataStore()
```

Read-only property that returns an object representing the currently connected *DataStore*, or **null** if no connection is open.

fileName

```
public final String getFileName()
public final synchronized void setFileName(String fileName)
```

Specifies the String name of the *DataStore* file to connect to. If the *DataStoreConnection* is open when the *setFileName()* method is called, a *DataSetException* is thrown.

If the *DataStore*'s *tempDirName* property is not set, temporary files are created in the directory location of *fileName*.

locale

```
public Locale getLocale()
```

Returns the locale associated with the connected *DataStore*. To set the locale, you must access the *locale* property of the *DataStore*.

This is the default locale setting for *StorageDataSet* objects that do not have a locale setting.

lockWaitTime

```
public final long getLockWaitTime()
public final void setLockWaitTime(long waitTime)
```

Determines the amount of time, in milliseconds, that the connection waits for a lock before aborting the current transaction. This is also used for deadlock detection. Setting this property to zero causes the connection to wait indefinitely. The default value is 10000 (10 seconds).

This property has no effect when the *DataStore* is not transactional.

open

```
public final boolean isOpen()
```

Read-only property that returns whether the connection to the *DataStore* is open (**true**) or not (**false**).

readOnlyTx

```
public final boolean isReadOnlyTx()
public final void setReadOnlyTx(boolean readOnlyTx)
```

Enables read-only mode on this connection. This property must be set before you open the connection. If **true**, when the connection opens, it will see a snapshot of all data, such that:

- It cannot write to the *DataStore*.
- Reads are not blocked by other transactions that are writing.
- Reads will only see data from committed transactions.

To refresh the snapshot of data, call the *commit* method for the connection (this starts a new read-only transaction).

readOnlyTxDelay

```
public final long getReadOnlyTxDelay()
public final void setReadOnlyTxDelay(long lagTime)
```

When the *readOnlyTx* property is **true**, the connection gets a snapshot of the DataStore data when it opens (or refreshes). This property specifies how long to hold onto a given snapshot, in milliseconds, so that it can be shared by other read-only connections. Sharing snapshots improves performance. Once the *lagTime* period ends, the next read-only transaction that opens (or refreshes) will get a new snapshot.

A value of 0 indicates there should be no delay, and that the connection should always check for the latest data.

storeInternals

```
public final StoreInternals getStoreInternals()
```

This property is used internally by other *com.borland* classes.

txIsolation

```
public final int getTxIsolation()
public final void setTxIsolation(int isolation)
```

Transaction isolation level for this connection. DataStore supports:

- TRANSACTION_SERIALIZABLE
- TRANSACTION_REPEATABLE_READ

as defined in *java.sql.connection*.

userName

```
public final String getUserName()
public final void setUserName(String userName)
```

The name of the user associated with this connection. The name is required when connecting to a transactional *DataStore*.

version

```
public final synchronized int getVersion()
```

Returns the *DataStore* file version, encoded in the following four bytes (from low byte to high byte):

Byte	Meaning
0	Not used
1	Major product version

Byte	Meaning
2	Minor product version
3	Minor file version

If no *DataStore* is open, a *DataStoreException.DATASTORE_NOT_OPEN* exception is thrown.

See also *getProductVersion* method

DataStoreConnection methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
closeDirectory()	this class
closeStream(java.lang.String)	this class
commit()	this class
copyStreams(java.lang.String, java.lang.String, com.borland.datastore.DataStoreConnection, java.lang.String, int, java.io.PrintStream)	this class
createFileStream(java.lang.String)	this class
deleteStream(java.lang.String)	this class
dropIndexStream(com.borland.dx.dataset.ReadRow)	this class
dumpLocks(java.io.PrintStream)	this class
dumpTxLog(java.io.PrintStream, long)	this class
equals(java.lang.Object)	java.lang.Object
fileExists(java.lang.String)	this class
finalize()	this class
getProductVersion()	this class
getStreamProperties(com.borland.dx.dataset.ReadRow)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	this class
openDirectory()	this class
openFileStream(java.lang.String)	this class
readObject(java.lang.String)	this class
renameStream(java.lang.String, java.lang.String)	this class
rollback()	this class
sortTable(java.lang.String, java.lang.String, com.borland.dx.dataset.SortDescriptor)	this class
tableExists(java.lang.String)	this class
toString()	java.lang.Object

Method	Implemented in
<code>transactionStarted()</code>	this class
<code>undeleteStream(com.borland.dx.dataset.ReadRow)</code>	this class
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>
<code>writeObject(java.lang.String, java.lang.Object)</code>	this class

close()

```
public void close()
```

Closes the connection to the *DataStore* component and flushes the data cache. All pending changes are committed. If it was the last connection to the *DataStore* the *DataStore* will shutdown.

On failure, this method throws a *DataSetException*. If an *IOException* is generated, it is returned in the chained *DataSetException* object.

closeDirectory()

```
public final synchronized void closeDirectory()
```

Closes the directory *StorageDataSet* (returned by the *openDirectory* method) that contains information on the directory structure of the *DataStore*'s file system.

In addition to releasing memory resources, closing the directory before directory update operations may improve their performance.

See also *openDirectory* method

closeStream(java.lang.String)

```
public final synchronized void closeStream(String storeName)
```

Forces the stream named *storeName* to close. Normally, table streams are closed through the accessing API (DataExpress or JDBC), and file streams are closed by calling the *close* method of the *FileStream* object returned by *createFileStream* or *openFileStream*. Calling *closeStream* bypasses these normal usage conventions and closes the stream at the source; further attempts to access the stream (without reopening new connections) will result in an appropriate exception.

commit()

```
public final boolean commit()
```

Commits all the changes made through the connection since the last *commit()* or *rollback()*.

Returns **true** if successful, or **false** if there was no work to commit, or if the connection's *readOnlyTx* property has been set.

copyStreams(java.lang.String, java.lang.String, com.borland.datastore.DataStoreConnection, java.lang.String, int, java.io.PrintStream)

```
public final void copyStreams(String sourceDir, String sourcePattern, DataStoreConnection destCon,
    String destDir, int options, PrintStream out)
```

Copies all or part of one *DataStore* to another. The copy operation will attempt to repair any damaged streams and copy them correctly. If a stream cannot be copied, *copyStreams* will either stop, or ignore the current stream and copy the rest (if any), depending on whether COPY_IGNORE_ERRORS is specified as one of the *options*.

Each stream is identified with a name that uses the "/" character as a directory separator to simulate a hierarchical directory structure. But *copyStreams* treats the name as a simple, single string. The *sourceDir* and *destDir* are useful primarily when changing directories during the copy; otherwise these two parameters should be specified as empty strings. The *sourcePattern* is compared against the part of the name that comes after the *sourceDir*; if *sourceDir* is empty, then *sourcePattern* is compared against the entire name.

If a stream already exists with the target name in the destination *DataStore*, the default action is to throw a *DataSetException*. You can choose to overwrite existing streams instead by specifying COPY_OVERWRITE as one of the *options*.

<i>sourceDir</i>	The directory to copy from. This directory will be replaced with the <i>destDir</i> name in the copy.
<i>sourcePattern</i>	The <i>storeName</i> of stream or streams to copy, using the standard "*" (multiple character) and "?" (single character) wild cards. By default, the name match is case-insensitive. You can make it case-sensitive by specifying COPY_CASE_SENSITIVE as one of the <i>options</i> .
<i>destCon</i>	A connection to the <i>DataStore</i> to copy to. You may specify a connection to the same <i>DataStore</i> , or even use the same <i>DataStoreConnection</i> , as long as the <i>sourceDir</i> and <i>destDir</i> parameters are different to make a copy of a stream inside the same <i>DataStore</i> .
<i>destDir</i>	The directory to copy to. The name of the copied stream will be prefixed with this string (after stripping off the old <i>sourceDir</i>).
<i>options</i>	Any of the <i>DataStoreConneciton.COPY_*</i> options, or zero to use the default options.
<i>out</i>	Status messages are routed to this <i>PrintStream</i> (if not null).

createFileStream(java.lang.String)

public final synchronized FileStream createFileStream(String storeName)

Creates a new *FileStream* inside the *DataStore* using *storeName* for the directory entry.

See also *openFileStream* method

deleteStream(java.lang.String)

public final synchronized void deleteStream(String storeName)

Deletes the named file from the *DataStore*. To restore the file (if possible), use the *undeleteFile()* method.

dropIndexStream(com.borland.dx.dataset.ReadRow)

public final synchronized boolean dropIndexStream(ReadRow dirEntry)

Deletes a secondary index stream (*DataStore.SECOND_INDEX_STREAM*), associated with a table stream (*DataStore.TABLE_STREAM*).

dirEntry The directory entry for the index stream to delete. The *DataStore.DIR_TYPE* column should be of type *DataStore.SECOND_INDEX_STREAM*.

dumpLocks(java.io.PrintStream)

public final synchronized void dumpLocks(PrintStream out)

Displays information for all locks current held by the connection to the designated output.

dumpTxLog(java.io.PrintStream, long)

public final synchronized void dumpTxLog(PrintStream out, long startLsn)

Displays the contents of *DataStore*'s transactional log file in text format.

out Destination for output.

startLsn Starting log sequence number. Use 0 to start at the earliest point in the current set of logs.

fileExists(java.lang.String)

public final boolean fileExists(String storeName)

Returns **true** if a file stream with the specified *storeName* exists, **false** if it doesn't exist. On error this method throws a *DataSetException*.

See also *tableExists* method

finalize()

```
public void finalize()
```

Closes the *DataStore* connection if it is still open.

Overrides `java.lang.Object.finalize()`

getProductVersion()

```
public static final String getProductVersion()
```

Returns the version of the *DataStore* package; not to be confused with the *version* property, which is the version of the open *DataStore* file.

See also *version* property

getStreamProperties(com.borland.dx.dataset.ReadRow)

```
public final synchronized StreamProperties getStreamProperties(ReadRow dirEntry)
```

Returns a *StreamProperties* object for a stream. On error, this method throws a *DataSetException*.

dirEntry The directory entry for the desired stream.

See also *StreamProperties* class

Example This code fragment scans through the streams in a *DataStore*, looking for an index stream with a particular name.

```
String indexName = "NameToFind";

TableDataSet storeDirectory = (TableDataSet) dataStore.openDirectory();

// This creates the structure of the row only
DataRow dirEntry = new DataRow(storeDirectory);

// Scan directory for secondary index entry
while (storeDirectory.inBounds()) {
    if (storeDirectory.getShort(DataStore.DIR_TYPE) == DataStore.SECOND_INDEX_STREAM) {
        // Copy the actual data to the DataRow
        storeDirectory.copyTo(dirEntry);
        streamProperties = dataStore.getStreamProperties(dirEntry);
        Diagnostic.println("IndexName=" + streamProperties.getIndexName());
        if (streamProperties.getIndexName().equals(indexName)) {
            break;
        }
    }
    storeDirectory.next();
}

Diagnostic.println("Found=" + streamProperties.getIndexName());
```

open()

```
public final synchronized void open()
```

Opens a connection to a *DataStore* using the *fileName* property to locate and open the *DataStore*. If the *DataStore* is transactional, then the *userName* property must also be set before opening. When a *StorageDataSet* that uses a *DataStore* is opened, it automatically asks the *DataStore* to open. This method generates a *DataStoreException* of *DATASTORE_NOT_FOUND* or an *IOException* if appropriate.

See also *close* method, *DataStore.create* method

openDirectory()

```
public final synchronized StorageDataSet openDirectory()
```

Returns information on the directory structure for this *DataStore* as a list of table and file streams in the *DataStore*. This information is returned as rows in a *StorageDataSet* component. On failure, this method throws a *DataSetException*.

The returned *StorageDataSet* is sorted by the first five of its columns as follows:

Column	Type	Description
State	short	Describes the state of the stream, comprising a valid combination of one or more <i>DataStore.*_STATE</i> values.
DeleteTime	long	UTC time a stream was deleted. If the stream has not been deleted, this value is always 0.
StoreName	String	Name for the stream, 1 to 192 bytes long.
Type	short	Describes the stream type, one of the <i>DataStore.*_STREAM</i> values.
Id	int	A unique identifier for a stream.
Properties	String	Lists persisted properties/events for a <i>DataSet</i> stream.
ModTime	long	UTC time the stream was last modified.
Length	long	Length of the stream in bytes.
BlobLength	long	Length of a table stream's BLOBs, in bytes.

The names of these columns are defined as *DIR_** constants in the *DataStore* class; for example, *DIR_STATE* is the string "State". Using these constants provides compile-time checking that the proper column name has been used.

Example This code fragment uses the directory returned by *openDirectory* to undelete a stream.

```
TableDataSet storeDirectory = (TableDataSet) datastore.openDirectory();

// Before deleting
Diagnostic.println("Stream exists: " + datastore.fileExists(streamName));
datastore.deleteStream(streamName);
Diagnostic.println("Stream exists: " + datastore.fileExists(streamName));

// This creates the structure of the row only
DataRow dirEntry = new DataRow(storeDirectory);

// Find the entry in the DataSet
dirEntry.setString(DataStore.DIR_STORE_NAME, streamName.toUpperCase());
storeDirectory.locate(dirEntry, Locate.FIRST);

// And copy the actual data to a new DataRow
storeDirectory.copyTo(dirEntry);
Diagnostic.println("Undeleting=" + storeDirectory.getString(DataStore.DIR_STORE_NAME));
datastore.undeleteStream(dirEntry);
Diagnostic.println("Stream exists: " + datastore.fileExists(streamName));
```

See also *closeDirectory* method

openFileStream(java.lang.String)

```
public final synchronized FileStream openFileStream(String storeName)
```

Opens an existing *FileStream*. Although *FileStream* extends *java.io.InputStream*, it also provides *seek()* and *write()* methods. This method throws a *DataStoreException* of *FILE_NOT_FOUND* if the stream does not exist.

See also *createFileStream* method

readObject(java.lang.String)

```
public final synchronized Object readObject(String storeName)
```

Retrieves a Java Object from a *DataStore* using Java serialization. The Object must implement *java.io.Serializable*; and its class must be accessible, or a *ClassNotFoundException* will be thrown.

See also *writeObject* method

renameStream(java.lang.String, java.lang.String)

```
public final void renameStream(String storeName, String newStoreName)
```

Renames a stream from *storeName* to *newStoreName*. If there is no stream with the original name *storeName*, nothing happens.

rollback()

public final boolean rollback()

Rolls back all the changes made through the connection since the last *commit()* or *rollback()*.

Returns **true** if successful, or **false** if there was no work to roll back, or if the connection's *readOnlyTx* property has been set.

sortTable(java.lang.String, java.lang.String, com.borland.dx.dataset.SortDescriptor)

public void sortTable(String source, String dest, SortDescriptor descriptor)

Makes a new sorted copy of a table stream. Note that the order of the new sorted table is not automatically maintained when it is edited; it's simply a new table that happens to have its rows in a particular order. To maintain the sort order, use the *sort* property of the *DataSet*.

source The *storeName* of the table to sort and copy.

dest The *storeName* of the new table to create.

sortDescriptor Describes the columns, order, and case-sensitivity of the sort.

tableExists(java.lang.String)

public final boolean tableExists(String storeName)

Returns **true** if a table stream with the specified *storeName* exists.

See also *fileExists* method

transactionStarted()

public final boolean transactionStarted()

Returns **true** if a transaction has been started; that is, if any changes have been made since the last *commit()* or *rollback()*.

undeleteStream(com.borland.dx.dataset.ReadRow)

public final synchronized boolean undeleteStream(ReadRow dirEntry)

Undeletes a deleted stream if its space has not been reallocated to other active streams. On error, this method throws a *DataSetException*.

dirEntry The directory entry for the stream to (attempt to) undelete.

writeObject(java.lang.String, java.lang.Object)

public final synchronized void writeObject(String storeName, Object object)

Stores a Java Object into the *DataStore* (using Java serialization) as a file stream with the name *storeName*. The Object must implement *java.io.Serializable*.

See also *readObject* method

DataStoreException class

datastore package

Extends com.borland.dx.dataset.DataSetException

Implements com.borland.jb.util.ChainedException, java.io.Serializable
DataStoreException enumerates DataStore-specific exceptions.
The *BASE* value for the class variables is 2000.

DataStoreException variables

Variable	Defined in
AGG_OPERATOR_NOT_FOUND	this class
ALREADY_LOADING	com.borland.dx.dataset.DataSetException
BAD_PROCEDURE_PROPERTIES	com.borland.dx.dataset.DataSetException
BAD_QUERY_PROPERTIES	com.borland.dx.dataset.DataSetException
BIGDECIMAL_PRECISION_ERROR	this class
CANNOT_CHANGE_COLUMN	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN_DATA_TYPE	com.borland.dx.dataset.DataSetException
CANNOT_FIND_TABLE_NAME	com.borland.dx.dataset.DataSetException
CANNOT_IMPORT_NULL_DATASET	com.borland.dx.dataset.DataSetException
CANNOT_OPEN	this class
CANNOT_REFRESH	com.borland.dx.dataset.DataSetException
CANNOT_RESTRUCTURE	this class
CANNOT_SAVE_CHANGES	com.borland.dx.dataset.DataSetException
CANNOT_UPDATE_SCOPED_DATA_ROW	com.borland.dx.dataset.DataSetException
CANT_CREATE_OPEN_STREAM	this class
CLASS_NOT_FOUND_ERROR	com.borland.dx.dataset.DataSetException
COLUMN_ALREADY_BOUND	com.borland.dx.dataset.DataSetException
COLUMN_NEEDS_RESTRUCTURE	this class
COLUMN_NOT_IN_ROW	com.borland.dx.dataset.DataSetException

Variable	Defined in
COLUMN_TYPE_CONFLICT	com.borland.dx.dataset.DataSetException
CONNECTION_DESCRIPTOR_NOT_SET	com.borland.dx.dataset.DataSetException
CONNECTION_NOT_CLOSED	com.borland.dx.dataset.DataSetException
DATA_FILE_LOAD_FAILED	com.borland.dx.dataset.DataSetException
DATA_HAS_DUPLICATES	this class
DATASET_ALREADY_OPEN	this class
DATASET_CORRUPT	com.borland.dx.dataset.DataSetException
DATASET_EXISTS	this class
DATASET_HAS_NO_ROWS	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_TABLES	com.borland.dx.dataset.DataSetException
DATASET_NOT_OPEN	com.borland.dx.dataset.DataSetException
DATASET_OPEN	com.borland.dx.dataset.DataSetException
DATASTORE_ALREADY_OPEN	this class
DATASTORE_EXISTS	this class
DATASTORE_INVALID	this class
DATASTORE_NOT_FOUND	this class
DATASTORE_NOT_OPEN	this class
DATASTORE_OPEN	this class
DELETE_DUPLICATES	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_AT_RUNTIME	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_IN_DESIGN	com.borland.dx.dataset.DataSetException
DUPLICATE_COLUMN_NAME	com.borland.dx.dataset.DataSetException
DUPLICATE_KEY	this class
DUPLICATE_PRIMARY	com.borland.dx.dataset.DataSetException
EMPTY_COLUMN_NAMES	com.borland.dx.dataset.DataSetException
errorCode	com.borland.dx.dataset.DataSetException
EXCEPTION_CHAIN	com.borland.dx.dataset.DataSetException
exceptionChain	com.borland.dx.dataset.DataSetException
FIELD_POST_ERROR	com.borland.dx.dataset.DataSetException
FILE_EXISTS	this class
GENERIC_ERROR	com.borland.dx.dataset.DataSetException
INCOMPATIBLE_BLOCK_SIZE	this class
INCOMPATIBLE_DATA_ROW	com.borland.dx.dataset.DataSetException
INLINE_TOO_SMALL	this class
INSUFFICIENT_ROWID	com.borland.dx.dataset.DataSetException
INVALID_AGG_DESCRIPTOR	com.borland.dx.dataset.DataSetException
INVALID_CLASS	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_POSITION	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_TYPE	com.borland.dx.dataset.DataSetException
INVALID_DATA_FILE_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_DIRECTORY_ATTRIBUTES	this class

Variable	Defined in
INVALID_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_ITERATOR_USE	com.borland.dx.dataset.DataSetException
INVALID_PATTERN	this class
INVALID_SCHEMA_FILE	com.borland.dx.dataset.DataSetException
INVALID_SORT_COLUMN	com.borland.dx.dataset.DataSetException
INVALID_STORE_CLASS	com.borland.dx.dataset.DataSetException
INVALID_STORE_NAME	com.borland.dx.dataset.DataSetException
IO_ERROR	com.borland.dx.dataset.DataSetException
LINK_COLUMNS_ERROR	com.borland.dx.dataset.DataSetException
LINKFIELD_IN_USERPARAMETERS	com.borland.dx.dataset.DataSetException
LOADING_NOT_STARTED	com.borland.dx.dataset.DataSetException
MASTER_DETAIL_VIEW_ERROR	com.borland.dx.dataset.DataSetException
MASTER_NAVIGATION_ERROR	com.borland.dx.dataset.DataSetException
MISMATCH_PARAM_RESULT	com.borland.dx.dataset.DataSetException
MISMATCHED_PARAMETER_FORMAT	com.borland.dx.dataset.DataSetException
MISSING_MASTER_DATASET	com.borland.dx.dataset.DataSetException
MISSING_REPLACESTOREROW	com.borland.dx.dataset.DataSetException
MISSING_RESOLVER	com.borland.dx.dataset.DataSetException
MULTIPLE_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NAME_NOT_UNIQUE	this class
NEED_LOCATE_START_OPTION	com.borland.dx.dataset.DataSetException
NEED_PROCEDUREPROVIDER	com.borland.dx.dataset.DataSetException
NEED_QUERYPROVIDER	com.borland.dx.dataset.DataSetException
NEED_STORAGEDATASET	com.borland.dx.dataset.DataSetException
NEEDS_RECALC	com.borland.dx.dataset.DataSetException
NEWER_VERSION	this class
NO_CALC_AGG_FIELDS	com.borland.dx.dataset.DataSetException
NO_CALC_FIELDS	com.borland.dx.dataset.DataSetException
NO_DATABASE_TO_RESOLVE	com.borland.dx.dataset.DataSetException
NO_NON_BLOB_COLUMNS	com.borland.dx.dataset.DataSetException
NO_PRIMARY_KEY	com.borland.dx.dataset.DataSetException
NO_PRIOR_ORIGINAL_ROW	com.borland.dx.dataset.DataSetException
NO_RESULT_SET	com.borland.dx.dataset.DataSetException
NO_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NO_UPDATABLE_COLUMNS	com.borland.dx.dataset.DataSetException
NO_WHERE_CLAUSE	com.borland.dx.dataset.DataSetException
NON_EXISTENT_ROWID	com.borland.dx.dataset.DataSetException
NOT_DATABASE_RESOLVER	com.borland.dx.dataset.DataSetException
NOT_SELECT_QUERY	com.borland.dx.dataset.DataSetException
NOT_UPDATEABLE	com.borland.dx.dataset.DataSetException
NULL_COLUMN_NAME	com.borland.dx.dataset.DataSetException

Variable	Defined in
OLDER_VERSION	this class
ONEPASS_INPUT_STREAM	com.borland.dx.dataset.DataSetException
OPERATION_CANCELED	this class
PARAMETER_COUNT_MISMATCH	com.borland.dx.dataset.DataSetException
PARTIAL_SEARCH_FOR_STRING	com.borland.dx.dataset.DataSetException
PROCEDURE_FAILED	com.borland.dx.dataset.DataSetException
PROCEDURE_IN_PROCESS	com.borland.dx.dataset.DataSetException
PROVIDER_FAILED	com.borland.dx.dataset.DataSetException
PROVIDER_OWNED	com.borland.dx.dataset.DataSetException
QUERY_FAILED	com.borland.dx.dataset.DataSetException
QUERY_IN_PROCESS	com.borland.dx.dataset.DataSetException
READ_BLOCK_ERROR	this class
READ_ONLY	this class
READ_ONLY_STORE	com.borland.dx.dataset.DataSetException
REFRESHROW_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
REOPEN_FAILURE	com.borland.dx.dataset.DataSetException
RESOLVE_FAILED	com.borland.dx.dataset.DataSetException
RESOLVE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
RESTRUCTURE_DATA_LOSS	this class
RESTRUCTURE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
RESTRUCTURE_PARSE_ERROR	this class
RESTRUCTURE_PRECISION_LOSS	this class
ROW_NOT_FOUND	this class
ROW_TOO_WIDE	this class
SET_CALCULATED_FAILURE	com.borland.dx.dataset.DataSetException
SQL_ERROR	com.borland.dx.dataset.DataSetException
STORE_NAME_NOT_SET	this class
STORE_OPERATION_UNSUPPORTED	this class
STREAM_CLOSED	this class
STREAM_NOT_FOUND	this class
STREAM_OPEN_TWICE	this class
TOO_MANY_ERRORS	this class
TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
UNEXPECTED_END_OF_QUERY	com.borland.dx.dataset.DataSetException
UNKNOWN_COLUMN_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_DETAIL_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_PARAM_NAME	com.borland.dx.dataset.DataSetException
UNRECOGNIZED_DATA_TYPE	com.borland.dx.dataset.DataSetException
UPDATE_FAILED	this class
URL_NOT_FOUND	com.borland.dx.dataset.DataSetException

Variable	Defined in
URL_NOT_FOUND_IN_DESIGN	com.borland.dx.dataset.DataSetException
WRITE_BLOCK_ERROR	this class
WRONG_DATABASE	com.borland.dx.dataset.DataSetException

AGG_OPERATOR_NOT_FOUND

public static final int AGG_OPERATOR_NOT_FOUND = BASE+24

The *com.borland.dx.dataset.AggOperator* class could not be loaded. Check the classpath setting for your application.

BIGDECIMAL_PRECISION_ERROR

public static final int BIGDECIMAL_PRECISION_ERROR = BASE+29

The precision of a *BigDecimal* value was exceeded.

CANNOT_OPEN

public static final int CANNOT_OPEN = BASE+47

Could not open the DataStore file specified in the *fileName* property. Make sure you have permission to access the file.

CANNOT_RESTRUCTURE

public static final int CANNOT_RESTRUCTURE = BASE+8

Cannot restructure when a table is opened by more than one *StorageDataSet*.

CANT_CREATE_OPEN_STREAM

public static final int CANT_CREATE_OPEN_STREAM = BASE+32

Cannot create or delete a stream that is still open.

COLUMN_NEEDS_RESTRUCTURE

public static final int COLUMN_NEEDS_RESTRUCTURE = BASE+9

The *Column* component is new or its data type has changed. Restructure the *DataSet*.

DATA_HAS_DUPLICATES

public static final int DATA_HAS_DUPLICATES = BASE+44

An attempt to create a unique index failed because the table had duplicate row values for the given column combination.

DATASET_ALREADY_OPEN

```
public static final int DATASET_ALREADY_OPEN = BASE+7
```

Attempt to open table with a *StorageDataSet* that has a different set of columns.

DATASET_EXISTS

```
public static final int DATASET_EXISTS = BASE+35
```

Cannot create a file stream with the same name as an existing table stream.

DATASTORE_ALREADY_OPEN

```
public static final int DATASTORE_ALREADY_OPEN = BASE+30
```

DataStore is already open by another process. Multiple connections are allowed only from the same process.

DATASTORE_EXISTS

```
public static final int DATASTORE_EXISTS = BASE+17
```

DataStore already exists; specify a name that does not already exist.

DATASTORE_INVALID

```
public static final int DATASTORE_INVALID = BASE+12
```

File specified by *fileName* property does not appear to be a valid DataStore file. The file signature or size is invalid.

DATASTORE_NOT_FOUND

```
public static final int DATASTORE_NOT_FOUND = BASE+11
```

Could not find the DataStore file specified in the *fileName* property, or the property is **null**. Make sure the property is set to the name of an existing file.

DATASTORE_NOT_OPEN

```
public static final int DATASTORE_NOT_OPEN = BASE+10
```

Operation failed. The *DataStore* is not open.

DATASTORE_OPEN

```
public static final int DATASTORE_OPEN = BASE+19
```

Operation failed. Operation cannot be performed on an open *DataStore*.

DUPLICATE_KEY

```
public static final int DUPLICATE_KEY = BASE+4
```

Operation failed. Attempt to add duplicate key value. This is different than a *ValidationException.DUPLICATE_KEY* error; this error is an internal unexpected key violation, not a unique constraint violation.

FILE_EXISTS

```
public static final int FILE_EXISTS = BASE+36
```

Cannot create a *DataSet* stream with the same name as an existing file stream.

INCOMPATIBLE_BLOCK_SIZE

```
public static final int INCOMPATIBLE_BLOCK_SIZE = BASE+46
```

Incompatible *DataStore.blockSize* property for *copyStream* operation. When copying between two *DataStores*, both *DataStores* must have the same *DataStore.BlockSize* property setting. (This requirement may be removed in the future.)

INLINE_TOO_SMALL

```
public static final int INLINE_TOO_SMALL = BASE+42
```

Column.MaxInline property setting is too small.

INVALID_DIRECTORY_ATTRIBUTES

```
public static final int INVALID_DIRECTORY_ATTRIBUTES = BASE+48
```

Cannot open the *DataSet* because it has invalid directory attributes.

This is an unexpected internal error symptomatic of a corrupted *DataStore* file. It is recommended that you use the *DataStore Explorer* or the *DataStoreConnection.copyStreams* method to save all other streams to a new *DataStore*. See the section on verifying and repairing *DataStores* in the documentation for the *DataStore* component.

INVALID_PATTERN

```
public static final int INVALID_PATTERN = BASE+45
```

Invalid escape sequence in string pattern; the escape character ‘\’ can only precede ‘*’, ‘?’ or ‘\’

NAME_NOT_UNIQUE

```
public static final int NAME_NOT_UNIQUE = BASE+15
```

The name of the storage file or table being added or renamed is not unique.

NEWER_VERSION

```
public static final int NEWER_VERSION = BASE+34
```

The DataStore file cannot be opened because it was created by a newer a version of the *DataStore*.

OLDER_VERSION

```
public static final int OLDER_VERSION = BASE+41
```

Cannot perform operation on DataStore because it was created with an older version of *DataStore*. To upgrade the DataStore file, use *DataStoreConnection.copyStreams()*.

OPERATION_CANCELED

```
public static final int OPERATION_CANCELED = BASE+25
```

Operation canceled. Used for canceling long running operations like sorting or restructuring.

READ_BLOCK_ERROR

```
public static final int READ_BLOCK_ERROR = BASE+23
```

Error reading from the *DataStore*. Unexpected block contents.

This is an unexpected internal error symptomatic of a corrupted DataStore file. It is recommended that you use the DataStore Explorer or the *DataStoreConnection.copyStreams* method to save all other streams to a new DataStore. See the section on verifying and repairing DataStores in the documentation for the *DataStore* component.

READ_ONLY

```
public static final int READ_ONLY = BASE+40
```

Write operation failed because the *DataStore* is read only.

RESTRUCTURE_DATA_LOSS

```
public static final int RESTRUCTURE_DATA_LOSS = BASE+26
```

Restructure operation is converting from one data type to another. Old values will not be converted to the new data type. See the Data type coercions table in the documentation for the *DataStore* component.

RESTRUCTURE_PARSE_ERROR

```
public static final int RESTRUCTURE_PARSE_ERROR = BASE+28
```

Restructure operation is converting from one data type to another. One or more parse errors occurred converting a string data type to a non-string data type.

RESTRUCTURE_PRECISION_LOSS

```
public static final int RESTRUCTURE_PRECISION_LOSS = BASE+27
```

Restructure operation is converting from one data type to another. Type conversion may result in precision loss when values of the old data type are converted to values of the new data type.

ROW_NOT_FOUND

```
public static final int ROW_NOT_FOUND = BASE+38
```

Unexpected condition. Internal row not found.

This is an unexpected internal error symptomatic of a corrupted DataStore file. It is recommended that you use the DataStore Explorer or the *DataStoreConnection.copyStreams* method to save all other streams to a new DataStore. See the section on verifying and repairing DataStores in the documentation for the *DataStore* component.

ROW_TOO_WIDE

```
public static final int ROW_TOO_WIDE = BASE+6
```

Maximum row size exceeded. Too many columns total, or too many with a high *Column.maxInline* property setting. There are two ways to remedy this problem:

- Set the *Column.maxInline* property for columns with the data types *Variant.STRING*, *Variant.INPUTSTREAM*, and *Variant.OBJECT* to a value less than 16. By default, the maximum inline storage for such fields will not go below 16 unless the *Column.maxInline* property is set lower.
- Recreate your DataStore with a larger block size. The *DataStore.blockSize* property must be set before the DataStore file is created. If you want to preserve existing data, create a new DataStore with the larger block size and use the *copyStreams* method to copy the data.

STORE_NAME_NOT_SET

```
public static final int STORE_NAME_NOT_SET = BASE+1
```

Operation failed. The *storeName* property for Table component not set.

STORE_OPERATION_UNSUPPORTED

```
public static final int STORE_OPERATION_UNSUPPORTED = BASE+16
```

Unexpected internal condition. Operation not supported on this stream.

STREAM_CLOSED

```
public static final int STREAM_CLOSED = BASE+31
```

InputStream from a *DataStore* has been closed and is no longer accessible.

STREAM_NOT_FOUND

```
public static final int STREAM_NOT_FOUND = BASE+33
```

Stream not found in *DataStore*.

STREAM_OPEN_TWICE

```
public static final int STREAM_OPEN_TWICE = BASE+14
```

Unexpected internal condition. Shut down and reopen the *DataStore*.

See also *DataStore.open()* method, *DataStore.shutdown()* method

TOO_MANY_ERRORS

```
public static final int TOO_MANY_ERRORS = BASE+39
```

Too many errors detected by the stream verifier.

UPDATE_FAILED

```
public static final int UPDATE_FAILED = BASE+13
```

Previous update failed. Shut down and reopen the *DataStore*.

See also *DataStore.open()* method, *DataStore.shutdown()* method

WRITE_BLOCK_ERROR

```
public static final int WRITE_BLOCK_ERROR = BASE+22
```

Error writing to the *DataStore* component. Unexpected cache block contents.

This is an unexpected internal error symptomatic of a corrupted *DataStore* file. It is recommended that you use the *DataStore Explorer* or the *DataStoreConnection.copyStreams* method to save all other streams to a new *DataStore*. See the section on verifying and repairing *DataStores* in the documentation for the *DataStore* component.

DataStoreException properties

Property	Implemented in
class*	java.lang.Object
errorCode*	com.borland.dx.dataset.DataSetException
exceptionChain*	com.borland.dx.dataset.DataSetException
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

DataStoreException methods

Method	Implemented in
addExceptionListener (com.borland.dx.dataset.ExceptionListener)	com.borland.dx.dataset.DataSetException
badProcedureProperties()	com.borland.dx.dataset.DataSetException
badQueryProperties()	com.borland.dx.dataset.DataSetException
classNotFoundException (java.lang.ClassNotFoundException)	com.borland.dx.dataset.DataSetException
clone()	java.lang.Object
connectionDescriptorNotSet()	com.borland.dx.dataset.DataSetException
connectionNotClosed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
dataSetHasNoTable()	com.borland.dx.dataset.DataSetException
dataSetNotOpen()	com.borland.dx.dataset.DataSetException
deleteDuplicates()	com.borland.dx.dataset.DataSetException
driverNotLoadedAtRuntime (java.lang.String)	com.borland.dx.dataset.DataSetException
driverNotLoadedInDesign (java.lang.String)	com.borland.dx.dataset.DataSetException
equals(java.lang.Object)	java.lang.Object
fillInStackTrace()	java.lang.Throwable
finalize()	java.lang.Object
getExceptionListeners()	com.borland.dx.dataset.DataSetException
hashCode()	java.lang.Object
insufficientRowId()	com.borland.dx.dataset.DataSetException
invalidClass(java.lang.Class)	com.borland.dx.dataset.DataSetException
invalidClass(java.lang.String, java.lang.String)	com.borland.dx.dataset.DataSetException
invalidColumnType (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
invalidSQLType(int)	com.borland.dx.dataset.DataSetException
invalidStoreName(java.lang.String)	com.borland.dx.dataset.DataSetException
IOException(java.io.IOException)	com.borland.dx.dataset.DataSetException

Method	Implemented in
mismatchedParameterFormat()	com.borland.dx.dataset.DataSetException
mismatchParamResult()	com.borland.dx.dataset.DataSetException
missingMasterDataSet()	com.borland.dx.dataset.DataSetException
mkUrlNotFound(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
mkUrlNotFoundInDesign(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
multipleRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
needProcedureProvider()	com.borland.dx.dataset.DataSetException
needQueryProvider()	com.borland.dx.dataset.DataSetException
needsRecalc(java.lang.String)	com.borland.dx.dataset.DataSetException
noDatabaseOnResolver()	com.borland.dx.dataset.DataSetException
nonExistentRowId()	com.borland.dx.dataset.DataSetException
noResultSet()	com.borland.dx.dataset.DataSetException
noRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
notDatabaseResolver()	com.borland.dx.dataset.DataSetException
notify()	java.lang.Object
notifyAll()	java.lang.Object
notSelectQuery()	com.borland.dx.dataset.DataSetException
notSortable()	com.borland.dx.dataset.DataSetException
noUpdatableColumns()	com.borland.dx.dataset.DataSetException
noWhereClause (com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.DataSetException
onePassInputStream (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
parameterCountMismatch(int, int, int)	com.borland.dx.dataset.DataSetException
printStackTrace()	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintStream)	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
procedureFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
procedureInProcess()	com.borland.dx.dataset.DataSetException
providerFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
providerOwned()	com.borland.dx.dataset.DataSetException
queryFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
queryInProcess()	com.borland.dx.dataset.DataSetException
readOnlyStore(java.lang.String)	com.borland.dx.dataset.DataSetException
removeExceptionListener(com.borland.dx. dataset.ExceptionListener)	com.borland.dx.dataset.DataSetException
resolveFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
SQLException(java.sql.SQLException)	com.borland.dx.dataset.DataSetException
throwException(int, java.lang.Exception)	com.borland.dx.dataset.DataSetException

Method	Implemented in
throwExceptionChain (java.lang.Throwable)	com.borland.dx.dataset.DataSetException
toString()	java.lang.Throwable
transactionIsolationLevelNotSupported()	com.borland.dx.dataset.DataSetException
unexpectedEndOfQuery()	com.borland.dx.dataset.DataSetException
unknownColumnName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownDetailName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownParamName(java.lang.String)	com.borland.dx.dataset.DataSetException
unrecognizedDataType()	com.borland.dx.dataset.DataSetException
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
wrongDatabase()	com.borland.dx.dataset.DataSetException

FileOutputStream class

datastore package

Extends java.io.OutputStream

The *FileOutputStream* provides read, write, and seek access for file streams stored in a *DataStore*.

See also *DataStore.open*

FileOutputStream properties

Property	Implemented in
class*	java.lang.Object

FileOutputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
flush()	java.io.OutputStream
hashCode()	java.lang.Object
notify()	java.lang.Object

Method	Implemented in
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(byte[], int, int)	this class
write(byte[])	java.io.OutputStream
write(int)	this class

close()

public final void close()

Closes the file stream. This releases resources held by the *DataStore*.

Overrides java.io.OutputStream.close()

write(byte[], int, int)

public void write(byte[] buf, int offset, int length)

Writes the specified *length* bytes from *buf* starting at *offset*.

Overrides java.io.OutputStream.write(byte[], int, int)

write(int)

public void write(int b)

Writes to the file stream.

Overrides java.io.OutputStream.write(int)

FileStream class

datastore package

Extends com.borland.datastore.BlobStream

Arbitrary files stored in a *DataStore* are accessed via the *FileStream* class. It provides full read/write/seek access to the file.

See also *DataStoreConnection.createFileStream()*, *DataStoreConnection.openFileStream()*

FileStream properties

Property	Implemented in
class*	java.lang.Object
open*	com.borland.datastore.BlobStream

FileStream methods

Method	Implemented in
available()	com.borland.datastore.BlobStream
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
mark(int)	com.borland.datastore.BlobStream
markSupported()	com.borland.datastore.BlobStream
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	com.borland.datastore.BlobStream
read(byte[], int, int)	com.borland.datastore.BlobStream
read(byte[])	java.io.InputStream
reset()	com.borland.datastore.BlobStream
seek(int)	com.borland.datastore.BlobStream
skip(long)	java.io.InputStream
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(byte[], int, int)	this class

close()

public void close()

Closes the file stream and releases resources held by the *DataStore*.

Overrides java.io.InputStream.close()

write(byte[], int, int)

```
public void write(byte[] buf, int offset, int length)
```

Writes the specified *length* bytes from *buf* starting at *offset* into the current position in the file stream.

StreamProperties class

datastore package

Extends java.lang.Object

The *StreamProperties* class contains properties for a *DataStore* stream. An instance of this class is returned using the *DataStore.getStreamProperties()* method.

StreamProperties properties

Property	Implemented in
caseInsensitive*	this class
class*	java.lang.Object
columns*	this class
descending*	this class
filterClassName*	this class
indexName*	this class
primary*	this class
unique*	this class

caseInsensitive

```
public final boolean isCaseInsensitive()
```

Read-only property that returns **true** if the stream type is *DataStore.SECOND_INDEX_STREAM* and it is a caseInsensitive secondary index.

columns

```
public final Column[] getColumns()
```

Read-only property that returns an array of Columns if this is a stream type of *DataStore.TABLE_STREAM* or *DataStore.SECOND_INDEX_STREAM*.

descending

```
public final boolean[] getDescending()
```

Read-only property that returns **true** if the stream type is *DataStore.SECOND_INDEX_STREAM* and it is a descending order secondary index.

filterClassName

```
public final String getFilterClassName()
```

Read-only property that returns the name of the class that filters the rows for this secondary index.

indexName

```
public final String getIndexName()
```

Read-only property that returns the index name if the stream type is *DataStore.SECOND_INDEX_STREAM*.

primary

```
public final boolean isPrimary()
```

This property returns **true** if the stream type is *DataStore.SECOND_INDEX_STREAM* and it is the primary index.

There is only one primary index and all columns in the primary index have a not-null constraint.

unique

```
public final boolean isUnique()
```

Returns **true** if the stream type is *DataStore.SECOND_INDEX_STREAM* and it is a unique secondary index.

StreamProperties methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
isDescending(int)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object

Method	Implemented in
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

isDescending(int)

public final boolean isDescending(int keyIndex)

Returns **true** if the stream type is *DataStore.SECOND_INDEX_STREAM* and its *keyIndex* column is in descending order.

StreamVerifier class

datastore package

Extends com.borland.datastore.BTree

The *StreamVerifier* class is used to verify the integrity of a file stream (*DataStore.FILE_STREAM*) or *DataSet* stream (*DataStore.TABLE_STREAM*). It defines several implementations for the *verify* method and various constants that customize the amount of status information returned from the verification process.

By default each *verify* method will check the stream(s), and either return the number of errors or throw an exception if the number of errors exceeds a specified number. You can use the *EXCEPTION* option to always throw an exception when an error is encountered.

See the section on verifying and repairing DataStores in the documentation for the *DataStore* component for more information.

StreamVerifier variables

Variable	Defined in
DATA	this class
EXCEPTION	this class
PROGRESS	this class
SILENT	this class
SILENT_VERBOSE	this class
SILENT_VERBOSE_EXCEPTION	this class
VERBOSE	this class

DATA

```
public static final int DATA = 0x0001
```

Constant that specifies to display the data in the stream.

EXCEPTION

```
public static final int EXCEPTION = 0x0004
```

Constant that specifies an *Exception* be thrown at the end of verification if any errors are encountered. By default an *Exception* is only thrown if more than the specified number of errors are encountered (as indicated in the *errorCount* parameter of all *verify(...)* methods.)

PROGRESS

```
public static final int PROGRESS = 0x0002
```

Constant that specifies to display the progress of stream verification.

SILENT

```
public static final int SILENT = 0x0008
```

Constant that instructs the verifier to not report any status information (that is, to ignore all display options specified in the *verify(...)* method) as long as there are no errors. If errors are encountered, the verification process is restarted using the specified display options.

SILENT_VERBOSE

```
public static final int SILENT_VERBOSE = SILENT|VERBOSE)
```

Constant that instructs the verifier to not report any status information (that is, to ignore all display options specified in the *verify(...)* method) as long as there are no errors. If errors are encountered, the verification process is restarted with maximum status information returned.

SILENT_VERBOSE_EXCEPTION

```
public static final int SILENT_VERBOSE_EXCEPTION = SILENT|VERBOSE|EXCEPTION)
```

Constant that instructs the verifier to not report any status information (that is, to ignore all display options specified in the *verify(...)* method) as long as there are no errors. If errors are encountered, the verification process is restarted with maximum status information returned and an *Exception* thrown at the end of the verification process.

VERBOSE

public static final int VERBOSE = DATA|PROGRESS)

Enables all status display options.

StreamVerifier properties

Property	Implemented in
class*	java.lang.Object

StreamVerifier methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.datastore.StreamProperties, com.borland.datastore.TxLogRef)	com.borland.datastore.BTree
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
typeName(short)	this class
verify(com.borland.datastore.DataStoreConnection, com.borland.dx.dataset.ReadRow, java.io.PrintStream, int, int)	this class
verify(com.borland.datastore.DataStoreConnection, java.io.PrintStream, int, int)	this class
verify(com.borland.dx.dataset.DataSet, java.io.PrintStream, int, int)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

typeName(short)

public static final String typeName(short type)

Converts the *DataStore* stream type into a readable string.

See also *AGG_STREAM, DELETED_STREAM, FETCH_STREAM, FILE_STREAM, HIDDEN_STREAM, INSERTED_STREAM, ORIGINALS_STREAM, SECOND_INDEX_STREAM, TABLE_FILE_STREAM, TABLE_STREAM*

verify(com.borland.datastore.DataStoreConnection, com.borland.dx.dataset.ReadRow, java.io.PrintStream, int, int)

```
public static final int verify(DataStoreConnection con, ReadRow dirEntry, PrintStream out, int displayOptions, int errorCount)
```

Checks the integrity of a single stream. Note that the stream is closed before verification begins. This method returns the number of errors found.

<i>con</i>	Connection to the DataStore that contains the <i>dirEntry</i> stream.
<i>dirEntry</i>	The stream to verify, designated by a row in a directory <i>DataSet</i> , returned by <i>DataStoreConnection.openDirectory()</i> .
<i>out</i>	The <i>PrintStream</i> to send the verification output to.
<i>displayOptions</i>	Instructs verifier how to display progress messages as the stream is verified.
<i>errorCount</i>	The number of errors that can be ignored before a <i>DataSetException</i> is thrown.

verify(com.borland.datastore.DataStoreConnection, java.io.PrintStream, int, int)

```
public static final int verify(DataStoreConnection con, PrintStream out, int displayOptions, int errorCount)
```

Checks the integrity of all the streams in the DataStore. Note that each stream is closed before verification begins. If no errors around found, the DataStore's *consistent* property is set to **true**. This method returns the number of errors found.

<i>con</i>	Connection to the DataStore to verify.
<i>out</i>	The <i>PrintStream</i> to send the verification output to.
<i>displayOptions</i>	Instructs verifier how to display progress messages as the stream is verified.
<i>errorCount</i>	The number of errors that can be ignored before a <i>DataSetException</i> is thrown.

verify(com.borland.dx.dataset.DataSet, java.io.PrintStream, int, int)

```
public static final int verify(DataSet dataSet, PrintStream out, int displayOptions, int errorCount)
```

Checks the integrity of all *DataStore.TABLE_STREAM* types used by the designated *DataSet*. This method returns the number of errors found.

<i>dataSet</i>	The <i>DataSet</i> streams to verify.
<i>out</i>	The <i>PrintStream</i> to send the verification output to.

<i>displayOptions</i>	Specifies how much status information is reported during the stream verification process. Valid values are defined as constants in this class.
<i>errorCount</i>	Indicates the number of errors that can be ignored before an <i>DataSetException</i> is thrown.

TxException class

datastore package

Extends	com.borland.dx.dataset.DataSetException
Implements	com.borland.jb.util.ChainedException, java.io.Serializable
	<i>TxException</i> enumerates transaction-specific DataStore exceptions.
	The <i>BASE</i> value for the class variables is 6000.

TxException variables

Variable	Defined in
ALREADY_LOADING	com.borland.dx.dataset.DataSetException
AMBIGUOUS_LOGDIR	this class
ANCHOR_TOO_OLD	this class
BAD_PROCEDURE_PROPERTIES	com.borland.dx.dataset.DataSetException
BAD_QUERY_PROPERTIES	com.borland.dx.dataset.DataSetException
BIRTHSTAMP_MISMATCH	this class
CANNOT_CHANGE_COLUMN	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN_DATA_TYPE	com.borland.dx.dataset.DataSetException
CANNOT_FIND_TABLE_NAME	com.borland.dx.dataset.DataSetException
CANNOT_IMPORT_NULL_DATASET	com.borland.dx.dataset.DataSetException
CANNOT_REFRESH	com.borland.dx.dataset.DataSetException
CANNOT_SAVE_CHANGES	com.borland.dx.dataset.DataSetException
CANNOT_UPDATE_SCOPED_DATA_ROW	com.borland.dx.dataset.DataSetException
CLASS_NOT_FOUND_ERROR	com.borland.dx.dataset.DataSetException
COLUMN_ALREADY_BOUND	com.borland.dx.dataset.DataSetException
COLUMN_NOT_IN_ROW	com.borland.dx.dataset.DataSetException
COLUMN_TYPE_CONFLICT	com.borland.dx.dataset.DataSetException
CONNECTION_DESCRIPTOR_NOT_SET	com.borland.dx.dataset.DataSetException
CONNECTION_NOT_CLOSED	com.borland.dx.dataset.DataSetException
DATA_FILE_LOAD_FAILED	com.borland.dx.dataset.DataSetException
DATASET_CORRUPT	com.borland.dx.dataset.DataSetException

Variable	Defined in
DATASET_HAS_NO_ROWS	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_TABLES	com.borland.dx.dataset.DataSetException
DATASET_NOT_OPEN	com.borland.dx.dataset.DataSetException
DATASET_OPEN	com.borland.dx.dataset.DataSetException
DELETE_DUPLICATES	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_AT_RUNTIME	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_IN_DESIGN	com.borland.dx.dataset.DataSetException
DUPLICATE_COLUMN_NAME	com.borland.dx.dataset.DataSetException
DUPLICATE_PRIMARY	com.borland.dx.dataset.DataSetException
EMPTY_COLUMN_NAMES	com.borland.dx.dataset.DataSetException
errorCode	com.borland.dx.dataset.DataSetException
EXCEPTION_CHAIN	com.borland.dx.dataset.DataSetException
exceptionChain	com.borland.dx.dataset.DataSetException
FIELD_POST_ERROR	com.borland.dx.dataset.DataSetException
GENERIC_ERROR	com.borland.dx.dataset.DataSetException
INCOMPATIBLE_DATA_ROW	com.borland.dx.dataset.DataSetException
INSUFFICIENT_ROWID	com.borland.dx.dataset.DataSetException
INVALID_AGG_DESCRIPTOR	com.borland.dx.dataset.DataSetException
INVALID_ANCHOR_FILE	this class
INVALID_ANCHOR_LENGTH	this class
INVALID_CLASS	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_POSITION	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_TYPE	com.borland.dx.dataset.DataSetException
INVALID_DATA_FILE_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_ITERATOR_USE	com.borland.dx.dataset.DataSetException
INVALID_LSN	this class
INVALID_SCHEMA_FILE	com.borland.dx.dataset.DataSetException
INVALID_SORT_COLUMN	com.borland.dx.dataset.DataSetException
INVALID_STORE_CLASS	com.borland.dx.dataset.DataSetException
INVALID_STORE_NAME	com.borland.dx.dataset.DataSetException
INVALID_USER_NAME	this class
IO_ERROR	com.borland.dx.dataset.DataSetException
LINK_COLUMNS_ERROR	com.borland.dx.dataset.DataSetException
LINKFIELD_IN_USERPARAMETERS	com.borland.dx.dataset.DataSetException
LOADING_NOT_STARTED	com.borland.dx.dataset.DataSetException
LOCK_TIMEOUT	this class
LOG_EXISTS	this class
LOGDIR_MISMATCH	this class
MASTER_DETAIL_VIEW_ERROR	com.borland.dx.dataset.DataSetException
MASTER_NAVIGATION_ERROR	com.borland.dx.dataset.DataSetException

Variable	Defined in
MISMATCH_PARAM_RESULT	com.borland.dx.dataset.DataSetException
MISMATCHED_PARAMETER_FORMAT	com.borland.dx.dataset.DataSetException
MISSING_CHECKPOINT_FILE	this class
MISSING_LOG_FILE	this class
MISSING_MASTER_DATASET	com.borland.dx.dataset.DataSetException
MISSING_REPLACESTOREROW	com.borland.dx.dataset.DataSetException
MISSING_RESOLVER	com.borland.dx.dataset.DataSetException
MULTIPLE_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NEED_A_LOGDIR	this class
NEED_LOCATE_START_OPTION	com.borland.dx.dataset.DataSetException
NEED_PROCEDUREPROVIDER	com.borland.dx.dataset.DataSetException
NEED_QUERYPROVIDER	com.borland.dx.dataset.DataSetException
NEED_STORAGEDATASET	com.borland.dx.dataset.DataSetException
NEEDS_RECALC	com.borland.dx.dataset.DataSetException
NO_CALC_AGG_FIELDS	com.borland.dx.dataset.DataSetException
NO_CALC_FIELDS	com.borland.dx.dataset.DataSetException
NO_DATABASE_TO_RESOLVE	com.borland.dx.dataset.DataSetException
NO_NON_BLOB_COLUMNS	com.borland.dx.dataset.DataSetException
NO_PRIMARY_KEY	com.borland.dx.dataset.DataSetException
NO_PRIOR_ORIGINAL_ROW	com.borland.dx.dataset.DataSetException
NO_RESULT_SET	com.borland.dx.dataset.DataSetException
NO_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NO_UPDATABLE_COLUMNS	com.borland.dx.dataset.DataSetException
NO_WHERE_CLAUSE	com.borland.dx.dataset.DataSetException
NON_EXISTENT_ROWID	com.borland.dx.dataset.DataSetException
NOT_DATABASE_RESOLVER	com.borland.dx.dataset.DataSetException
NOT_SELECT_QUERY	com.borland.dx.dataset.DataSetException
NOT_TRANSACTIONAL	this class
NOT_UPDATEABLE	com.borland.dx.dataset.DataSetException
NULL_COLUMN_NAME	com.borland.dx.dataset.DataSetException
OLD_LOG	this class
ONEPASS_INPUT_STREAM	com.borland.dx.dataset.DataSetException
PARAMETER_COUNT_MISMATCH	com.borland.dx.dataset.DataSetException
PARTIAL_SEARCH_FOR_STRING	com.borland.dx.dataset.DataSetException
PROCEDURE_FAILED	com.borland.dx.dataset.DataSetException
PROCEDURE_IN_PROCESS	com.borland.dx.dataset.DataSetException
PROVIDER_FAILED	com.borland.dx.dataset.DataSetException
PROVIDER_OWNED	com.borland.dx.dataset.DataSetException
QUERY_FAILED	com.borland.dx.dataset.DataSetException
QUERY_IN_PROCESS	com.borland.dx.dataset.DataSetException
READ_ONLY_STORE	com.borland.dx.dataset.DataSetException

Variable	Defined in
RECOVERY_DENIED	this class
REFRESHROW_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
REOPEN_FAILURE	com.borland.dx.dataset.DataSetException
RESOLVE_FAILED	com.borland.dx.dataset.DataSetException
RESOLVE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
RESTRUCTURE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
ROW_LOCK_TIMEOUT	this class
SET_CALCULATED_FAILURE	com.borland.dx.dataset.DataSetException
SQL_ERROR	com.borland.dx.dataset.DataSetException
TOO_MANY_OPEN_FILES	this class
TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
TX_MANAGER_INUSE	this class
TX_NOT_ALLOWED	this class
TX_REQUIRED	this class
UNEXPECTED_END_OF_QUERY	com.borland.dx.dataset.DataSetException
UNKNOWN_COLUMN_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_DETAIL_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_PARAM_NAME	com.borland.dx.dataset.DataSetException
UNRECOGNIZED_DATA_TYPE	com.borland.dx.dataset.DataSetException
URL_NOT_FOUND	com.borland.dx.dataset.DataSetException
URL_NOT_FOUND_IN_DESIGN	com.borland.dx.dataset.DataSetException
WRONG_DATABASE	com.borland.dx.dataset.DataSetException

AMBIGUOUS_LOGDIR

```
public static final int AMBIGUOUS_LOGDIR = BASE+5
```

Log directory setting persisted in the DataStore is not the same as the setting in the *TxManager*, and the directory specified by the DataStore still contains log files.

If the log files were copied to the new directory specified by the *TxManager*, the log files in the old location (persisted in the DataStore file) must be renamed or deleted.

ANCHOR_TOO_OLD

```
public static final int ANCHOR_TOO_OLD = BASE+9
```

Anchor timestamp is older then timestamp of DataStore file. This is probably because the log files are out of date with the DataStore file.

BIRTHSTAMP_MISMATCH

```
public static final int BIRTHSTAMP_MISMATCH = BASE+8
```

Birthstamp for “A” anchor file and “B” anchor file do not match. This probably means that the log files were not created for the same DataStore file.

INVALID_ANCHOR_FILE

```
public static final int INVALID_ANCHOR_FILE = BASE+7
```

Anchor file could not be read or has an invalid contents. It is probably corrupt, but this should be a rare error since the precious anchor is copied to at least two separate files. The system redundantly copies this anchor inside the DataStore file itself, so this could be a recoverable failure. Log duplexing is another way to recover from such a failure. If the log is being duplexed (by setting the *TxManager.BLogDir* property), the “A” and “B” anchors will be examined and the newest, valid anchor will be used.

INVALID_ANCHOR_LENGTH

```
public static final int INVALID_ANCHOR_LENGTH = BASE+6
```

Anchor file has an invalid length. It is probably corrupt, but this should be a rare error since the precious anchor is copied to at least two separate files. The system redundantly copies this anchor inside the DataStore file itself, so this could be a recoverable failure. Log duplexing is another way to recover from such a failure. If the log is being duplexed (by setting the *TxManager.BLogDir* property), the “A” and “B” anchors will be examined and the newest, valid anchor will be used.

INVALID_LSN

```
public static final int INVALID_LSN = BASE+15
```

Unexpected internal condition. Invalid address into a log file.

This is an unexpected internal error symptomatic of a corrupted log file. It is recommended that you use the DataStore Explorer or the *DataStoreConnection.copyStreams* method to save all other streams to a new DataStore. See the section on verifying and repairing DataStores in the documentation for the *DataStore* component.

INVALID_USER_NAME

```
public static final int INVALID_USER_NAME = BASE+11
```

Invalid user name for *DataStoreConnection.userName* property. Typically caused when the property has never been set.

LOCK_TIMEOUT

```
public static final int LOCK_TIMEOUT = BASE+24
```

Timeout waiting for lock. This can be caused by a deadlock scenario or because a connection with a lock is taking a long time to commit its transaction.

See also *DataStoreConnection.lockWaitTime* property

LOG_EXISTS

```
public static final int LOG_EXISTS = BASE+0
```

Existing log files and/or Log anchor file found in the directory where the system is attempting to create new log files.

LOGDIR_MISMATCH

```
public static final int LOGDIR_MISMATCH = BASE+4
```

“A” and “B” log directories specified for DataStore, but one of them does not exist. The log files must both exist or both not exist. This ensures that the duplexed logs are in sync.

MISSING_CHECKPOINT_FILE

```
public static final int MISSING_CHECKPOINT_FILE = BASE+17
```

Cannot open the DataStore. Missing the log file with the most recent checkpoint.

MISSING_LOG_FILE

```
public static final int MISSING_LOG_FILE = BASE+16
```

Missing log file.

NEED_A_LOGDIR

```
public static final int NEED_A_LOGDIR = BASE+3
```

“B” log directory specified without specifying “A” log directory. The “A” log directory must be specified. “B” log directory is an optional duplexing of the log files.

NOT_TRANSACTIONAL

```
public static final int NOT_TRANSACTIONAL = BASE+14
```

Attempted transactional operation on DataStore when its *txManager* property is **null**.

OLD_LOG

```
public static final int OLD_LOG = BASE+25
```

The log file appears to be older than the *DataStore* it is associated with. This may be caused by copying old log files to the directory of a *DataStore* database file that has more recent changes.

To set up new log files, you can open the *DataStore* in read-only mode and copy its contents. To open the *DataStore* in read-only mode, use *DataStoreConnection.setReadOnly()*. To copy its contents, use *DataStoreConnection.copyStreams()*.

RECOVERY_DENIED

```
public static final int RECOVERY_DENIED = BASE+22
```

Recovery denied by *ResponseListener* event handler. To prevent this exception and allow recovery to complete, do not deny this event in your application's *ResponseListener* handler.

See also *ResponseEvent.DATASTORE_RECOVERING*

ROW_LOCK_TIMEOUT

```
public static final int ROW_LOCK_TIMEOUT = BASE+26
```

Timeout waiting for row lock. This can be caused by a deadlock scenario or because a connection with a lock is taking a long time to commit the transaction that has acquired a lock.

See also *DataStoreConnection.lockWaitTime* property

TOO_MANY_OPEN_FILES

```
public static final int TOO_MANY_OPEN_FILES = BASE+10
```

Too many files opened simultaneously by the log manager. This should not happen. If this does happen, shutdown and restart the system.

TX_MANAGER_INUSE

```
public static final int TX_MANAGER_INUSE = BASE+18
```

Attempt to use a *TxManager()* component instance for two different *DataStores*.

TX_NOT_ALLOWED

```
public static final int TX_NOT_ALLOWED = BASE+21
```

Cannot make this *DataStore* transactional because it is an older file format.

TX_REQUIRED

public static final int TX_REQUIRED = BASE+20

This connection cannot be opened because the DataStore is neither transactional nor read-only. This error can be corrected by setting the *DataStore.readOnly* property to true or by setting the *DataStore.txManager* property to an instance of *TxManager*. The DataStore JDBC driver can only open connections to transactional or read-only DataStores.

Once a DataStore component is opened with the *txManager* property set, the DataStore file is marked transactional and all *TxManager* properties are persisted in the DataStore file. So this operation only needs to be performed once.

TxException properties

Property	Implemented in
class*	java.lang.Object
errorCode*	com.borland.dx.dataset.DataSetException
exceptionChain*	com.borland.dx.dataset.DataSetException
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

TxException methods

Method	Implemented in
addExceptionListener(com.borland.dx.dataset.ExceptionListener)	com.borland.dx.dataset.DataSetException
badProcedureProperties()	com.borland.dx.dataset.DataSetException
badQueryProperties()	com.borland.dx.dataset.DataSetException
classNotFoundException (java.lang.ClassNotFoundException)	com.borland.dx.dataset.DataSetException
clone()	java.lang.Object
connectionDescriptorNotSet()	com.borland.dx.dataset.DataSetException
connectionNotClosed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
dataSetHasNoTable()	com.borland.dx.dataset.DataSetException
dataSetNotOpen()	com.borland.dx.dataset.DataSetException
deleteDuplicates()	com.borland.dx.dataset.DataSetException
driverNotLoadedAtRuntime (java.lang.String)	com.borland.dx.dataset.DataSetException
driverNotLoadedInDesign (java.lang.String)	com.borland.dx.dataset.DataSetException
equals(java.lang.Object)	java.lang.Object

Method	Implemented in
fillInStackTrace()	java.lang.Throwable
finalize()	java.lang.Object
getExceptionListeners()	com.borland.dx.dataset.DataSetException
hashCode()	java.lang.Object
insufficientRowId()	com.borland.dx.dataset.DataSetException
invalidClass(java.lang.Class)	com.borland.dx.dataset.DataSetException
invalidClass(java.lang.String, java.lang.String)	com.borland.dx.dataset.DataSetException
invalidColumnType (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
invalidSQLType(int)	com.borland.dx.dataset.DataSetException
invalidStoreName(java.lang.String)	com.borland.dx.dataset.DataSetException
IOException(java.io.IOException)	com.borland.dx.dataset.DataSetException
mismatchedParameterFormat()	com.borland.dx.dataset.DataSetException
mismatchParamResult()	com.borland.dx.dataset.DataSetException
missingMasterDataSet()	com.borland.dx.dataset.DataSetException
mkUrlNotFound(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
mkUrlNotFoundInDesign(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
multipleRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
needProcedureProvider()	com.borland.dx.dataset.DataSetException
needQueryProvider()	com.borland.dx.dataset.DataSetException
needsRecalc(java.lang.String)	com.borland.dx.dataset.DataSetException
noDatabaseOnResolver()	com.borland.dx.dataset.DataSetException
nonExistentRowId()	com.borland.dx.dataset.DataSetException
noResultSet()	com.borland.dx.dataset.DataSetException
noRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
notDatabaseResolver()	com.borland.dx.dataset.DataSetException
notify()	java.lang.Object
notifyAll()	java.lang.Object
notSelectQuery()	com.borland.dx.dataset.DataSetException
notSortable()	com.borland.dx.dataset.DataSetException
noUpdatableColumns()	com.borland.dx.dataset.DataSetException
noWhereClause (com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.DataSetException
onePassInputStream (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
parameterCountMismatch(int, int, int)	com.borland.dx.dataset.DataSetException
printStackTrace()	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintStream)	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintWriter)	java.lang.Throwable

Method	Implemented in
procedureFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
procedureInProgress()	com.borland.dx.dataset.DataSetException
providerFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
providerOwned()	com.borland.dx.dataset.DataSetException
queryFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
queryInProgress()	com.borland.dx.dataset.DataSetException
readOnlyStore(java.lang.String)	com.borland.dx.dataset.DataSetException
removeExceptionListener(com.borland.dx.dataset.ExceptionListener)	com.borland.dx.dataset.DataSetException
resolveFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
SQLException(java.sql.SQLException)	com.borland.dx.dataset.DataSetException
throwException(int, java.lang.Exception)	com.borland.dx.dataset.DataSetException
throwExceptionChain (java.lang.Throwable)	com.borland.dx.dataset.DataSetException
toString()	java.lang.Throwable
transactionIsolationLevelNotSupported()	com.borland.dx.dataset.DataSetException
unexpectedEndOfQuery()	com.borland.dx.dataset.DataSetException
unknownColumnName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownDetailName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownParamName(java.lang.String)	com.borland.dx.dataset.DataSetException
unrecognizedDataType()	com.borland.dx.dataset.DataSetException
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
wrongDatabase()	com.borland.dx.dataset.DataSetException

TxManager component

datastore package

Extends java.lang.Object

Implements com.borland.dx.dataset.Designable, java.lang.Runnable

The *TxManager* enables transaction and crash recovery support for a *DataStore*. You may assign a *TxManager* to a *DataStore* before you *create()* or *open()* it. For example, if you want to use the default properties for a *TxManager*, all you need to make a *DataStore* transactional is:

```
dataStore.setTxManager( new TxManager() );
dataStore.open();
```

Whenever a *DataStore* is transaction-enabled the first time, it creates a number of log files. For greater reliability, you can choose to duplex the log files, creating two identical copies (usually in different locations). The names

of the log files use the name of the DataStore file, without the file extension. For example, if the DataStore file `MyStore.jds` uses simplex transaction logging, the following log files are created:

- The status file `MyStore_STATUS_0000000000`,
- The anchor file `MyStore_LOGA_ANCHOR`, and
- The record file `MyStore_LOGA_0000000000`.

Duplex logging adds the files `MyStore_LOGB_ANCHOR` and `MyStore_LOGB_0000000000`. Additional status and record files are created as needed, with the log file number incrementing by one each time. As old log files are no longer needed for active transactions or crash recovery, they are automatically deleted. Old log files can be saved by listening to the *DataStore.response* event for a *ResponseEvent.DROP_LOG* notification. At that point, you can copy out the log file to another location before it is deleted, or *cancel* the event to prevent the deletion of the log file.

The location of the log files is just one of several *TxManager* properties that can be chosen before assigning the *TxManager* to the DataStore. The *TxManager* properties are persisted inside the DataStore file and the `LOG?_ANCHOR` files. These properties can be changed by instantiating a new *TxManager* component, assigning the properties that you want to change (the properties you do not touch will not be changed), setting the *DataStore.txManager* property to the new *TxManager*, and reopening the DataStore.

Because the *TxManager* properties are persisted in the DataStore file, the DataStore will automatically instantiate, configure, and use a *TxManager* when it opens, if a *TxManager* is not assigned. You can open a transactional DataStore without a *TxManager* by setting the DataStore's *readOnly* property to **true** before opening it.

Note that the location of the log files is persisted with the full drive and path. For example, if you want to create a transactional DataStore for later deployment to a server, you should create it in the same drive and path on your development machine as the eventual location on the server.

Making a DataStore transactional increases the initial size of the file. This preallocation of disk clusters decreases the chance of write failures and improves the chances for data recovery.

The properties and methods of *TxManager* configure the operation of the transaction management engine. You do not commit and roll back transactions through the *TxManager*; these operations are part of the *DataStoreConnection* class.

TxManager variables

Variable	Defined in
DEFAULT_CHECK_FREQUENCY	this class
DEFAULT_LOG_BLOCK_SIZE	this class
DEFAULT_LOG_SIZE	this class
DEFAULT_OPEN_LOGS	this class
MAX_CHECK_FREQUENCY	this class
MAX_LOG_BLOCK_SIZE	this class
MAX_OPEN_LOGS	this class
MIN_CHECK_FREQUENCY	this class
MIN_LOG_BLOCK_SIZE	this class
MIN_LOG_SIZE	this class
MIN_OPEN_LOGS	this class

DEFAULT_CHECK_FREQUENCY

public static final int DEFAULT_CHECK_FREQUENCY = 1024 * 1024 * 2

Default value for the *checkFrequency* property; also used when invalid values are assigned.

DEFAULT_LOG_BLOCK_SIZE

public static final int DEFAULT_LOG_BLOCK_SIZE = 4

Default value for the *logBlockSize* property; also used when invalid values are assigned.

DEFAULT_LOG_SIZE

public static final int DEFAULT_LOG_SIZE = 1024*1024)*64

Default value for the *maxLogSize* property; also used when invalid values are assigned.

DEFAULT_OPEN_LOGS

public static final int DEFAULT_OPEN_LOGS = 2

Default value for the *maxOpenLogs* property; also used when invalid values are assigned.

MAX_CHECK_FREQUENCY

```
public static final int MAX_CHECK_FREQUENCY = Integer.MAX_VALUE
```

Maximum value for the *checkFrequency* property. If a larger value is assigned, the property reverts to *DEFAULT_CHECK_FREQUENCY*.

MAX_LOG_BLOCK_SIZE

```
public static final int MAX_LOG_BLOCK_SIZE = 16
```

Maximum value for the *logBlockSize* property. If a larger value is assigned, the property reverts to *DEFAULT_LOG_BLOCK_SIZE*.

MAX_OPEN_LOGS

```
public static final int MAX_OPEN_LOGS = 16
```

Maximum value for the *maxOpenLogs* property. If a larger value is assigned, the property reverts to *DEFAULT_OPEN_LOGS*.

MIN_CHECK_FREQUENCY

```
public static final int MIN_CHECK_FREQUENCY = 1024 * 8
```

Minimum value for the *checkFrequency* property. If a smaller value is assigned, the property reverts to *DEFAULT_CHECK_FREQUENCY*.

MIN_LOG_BLOCK_SIZE

```
public static final int MIN_LOG_BLOCK_SIZE = 4
```

Minimum value for the *logBlockSize* property. If a smaller value is assigned, the property reverts to *DEFAULT_LOG_BLOCK_SIZE*.

MIN_LOG_SIZE

```
public static final int MIN_LOG_SIZE = 1024*1024)
```

Minimum value for the *maxLogSize* property. If a smaller value is assigned, the property reverts to *DEFAULT_LOG_SIZE*.

MIN_OPEN_LOGS

```
public static final int MIN_OPEN_LOGS = 2
```

Minimum value for the *maxOpenLogs* property. If a smaller value is assigned, the property reverts to *DEFAULT_OPEN_LOGS*.

TxManager constructors

TxManager()

public TxManager()

Instantiates a *TxManager* with default property settings.

TxManager properties

Property	Implemented in
ALogDir	this class
BLogDir	this class
checkFrequency	this class
class*	java.lang.Object
enabled	this class
logBlockSize	this class
maxLogSize	this class
maxOpenLogs	this class
recordStatus	this class
softCommit	this class

ALogDir

public final String getALogDir()

public synchronized void setALogDir(String aLogDir)

Directory location of the primary copy of the log files, referred to as the “A” log files, and the `STATUS` file, which collects status messages. By default these are kept in the same directory as the DataStore file.

BLogDir

public final String getBLogDir()

public synchronized void setBLogDir(String bLogDir)

Directory location of the secondary backup copy of the log files, referred to as the “B” log files. By default these are kept in the same directory as the DataStore.

Duplexing the log files will decrease performance, but increase the probability of a successful crash recovery. The only time log duplexing will benefit crash recovery is when there is media damage to the “A” log files or the “A” log files are lost, and the “B” log files are not damaged in the same locations or lost. Therefore, it’s best to place the “B” log files on different media, preferably a different physical disk drive (which can also alleviate some of the performance penalty).

checkFrequency

```
public int getCheckFrequency()
public synchronized void setCheckFrequency(int checkFrequency)
```

Frequency at which checkpoints are made to the log. *checkFrequency* is the amount of log file to be generated before starting a new checkpoint. A smaller value provides for faster crash recovery and faster-growing log files while a larger value provides for slower crash recovery and slower-growing log files.

enabled

```
public final boolean isEnabled()
public final void setEnabled(boolean enabled)
```

Enables or disables transaction support. *enabled* is **true** by default. If a transactional *dataStore* is opened with its *txManager* property set to a *TxManager* that has *enabled* set to **false**, the *DataStore* will become non-transactional. This is useful for transporting a *DataStore* file when you do not want to transfer the associated log files. If the *DataStore* has its *txManager* property set back to a *TxManager* that is enabled, it will become transactional again.

Note that transaction support cannot be disabled for a *DataStore* when the *DataStore.consistent* property is **false**. In this case it is best to copy the contents of the *DataStore* to a new *DataStore* by using the *DataStore Explorer* or the *DataStoreConnection.copyStreams* method.

logBlockSize

```
public final int getLogBlockSize()
public final void setLogBlockSize(int logBlockSize)
```

Block size in increments of 1024 bytes to use for reading, writing, and caching of log file data.

maxLogSize

```
public int getMaxLogSize()
public synchronized void setMaxLogSize(int maxLogSize)
```

When a log file fills up to this size, a new log file (with the next number) is created and used. For example, after *MyStore_LOGA_0000000000* reaches this size, *MyStore_LOGA_0000000001* is created to record further transactions.

maxOpenLogs

```
public int getMaxOpenLogs()
public synchronized void setMaxOpenLogs(int maxOpenLogs)
```

Maximum number of log files that will be kept open at one time. Higher values may improve performance, but they will consume more system file handles.

recordStatus

```
public final boolean isRecordStatus()  
public final synchronized void setRecordStatus(boolean recordStatus)
```

Controls whether various status messages will be recorded to the `STATUS` log files, which are stored in the “A” log directory.

See also `setALogDir()`

softCommit

```
public boolean isSoftCommit()  
public synchronized void setSoftCommit(boolean softCommit)
```

Enables soft recovery.

When this property is set to **true**, crash recovery is still guaranteed, but the durability of the most recent committed transactions are not. Transactions committed within a second should be durable. Forcing (sync to disk) of the log file is performed for crash recovery purposes, but not to guarantee a transaction commit.

With *softCommit* set to **false** (the default) durability of transactions is guaranteed.

Setting this property improves performance, but can lead to recently committed transactions being rolled back after a system crash.

TxManager methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
run()	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

datastore.jdbc package

The *datastore.jdbc* package contains the JDBC interface for the DataStore, including the JDBC driver itself, and classes for implementing your own DataStore server for multi-user connections to the same DataStore.

The following classes, components, and interfaces in this package are intended for public use:

- DataStoreDriver
- DataStoreServer
- ServerConnection
- ServerStatus
- ServerStatusEvent
- ServerStatusListener

For more information, visit the database newsgroup. Details on newsgroups can be found at <http://www.borland.com/newsgroups>. The database newsgroup is dedicated to issues about writing database applications and is actively monitored by our support engineers as well as the Development team.

Interfaces

ServerStatus

ServerStatusListener

Classes and components

DataStoreDriver

DataStoreServer

ServerConnection

ServerStatusEvent

Overview of classes in the *com.borland.datastore.jdbc* package

<code>DataStoreDriver</code>	The JDBC driver for both local and remote/ multi-user connections to a <code>DataStore</code> .
<code>DataStoreServer</code>	Starts a thread that accepts multiple client connection requests to a <code>DataStore</code> from the remote <code>DataStore</code> JDBC driver. This class is the engine of any <code>DataStore</code> server implementation.
<code>ServerConnection</code>	A thread that represents connections to a <i>DataStoreServer</i> .
<code>ServerStatus</code>	An interface that defines possible server status events reported to <i>ServerStatusListeners</i> .
<code>ServerStatusEvent</code>	A status change, such as a connection or disconnection, or error that occurs in the server or one of its connections.
<code>ServerStatusListener</code>	An interface that receives <i>ServerEvent</i> events.

DataStoreDriver component

`datastore.jdbc` package

Extends `java.lang.Object`

Implements `java.sql.Driver`

DataStoreDriver is the JDBC driver for the *DataStore*. The driver supports both local and remote access. Both types of access require a user name (any string, with no setup required) and an empty password.

Local access provides exclusive, transactional, single-user access to a `DataStore` file that is accessible by the machine requesting the connection. Exclusive access means that the `DataStore` file cannot be opened by another process. You can open other connections from the same process, using another JDBC connection, the `DataExpress` API, or the `DataStore` API. The JDBC URL for local access is:

```
jdbc:borland:dslocal:<filename>
```

Remote access requires a `DataStore` server. By routing all requests through a single `DataStore` server process, you get transactional multi-user access to a `DataStore`. The `DataStore` file must be accessible from the `DataStore` server. The JDBC URL for remote access is:

```
jdbc:borland:dsremote://<hostname>/<filename>
```

For example, a simple JDBC local connection might look like:

```
Class.forName( "com.borland.datastore.jdbc.DataStoreDriver" );
Connection con = DriverManager.getConnection(
    "jdbc:borland:dslocal:c:/mydir/mystore.jds",
    "MyUserName", "" );
```

A remote connection using DataExpress might look like:

```
ConnectionDescriptor con = new ConnectionDescriptor(
    "jdbc:borland:dsremote://MYSERVER/c:/serverdir/serverdb.jds",
    "MyUserName", "", false,
    "com.borland.datastore.jdbc.DataStoreDriver" );
```

Driver properties

The DataStore JDBC driver supports the following properties:

Table 3.1 DataStore JDBC driver properties

Property name	Value (all strings)
user	Any name (no setup required)
password	Password. There is no security with <i>DataStoreDriver</i> , so this should be an empty string.
port	Port to connect on; the default is 2508.
readonly	"true" or "false": whether to open the connection read-only.
lockWaitTime	Time to wait for lock before aborting transaction, in milliseconds; default 10000 (10 seconds)

The following example gets a read-only connection on port 9876 using driver properties:

```
Class.forName( "com.borland.datastore.jdbc.DataStoreDriver" );
java.util.Properties info = new java.util.Properties();
info.setProperty( "user" , "MyUserName" );
info.setProperty( "password", "" );
info.setProperty( "port" , "9876" );
info.setProperty( "readonly", "true" );
Connection con = DriverManager.getConnection(
    "jdbc:borland:dsremote://MYSERVER/c:/serverdir/serverdb.jds", info );
```

SQL support

DataStoreDriver supports a subset of the ANSI/ISO SQL-92 standard. In general, it provides:

- Some Data Definition Language for managing tables and indexes, but no schema, domain, views, or security elements.
- Data manipulation and selection with INSERT, UPDATE, DELETE, and SELECT; but no cursors.
- Transaction support.
- No security support.

DataStoreDriver constructors

DataStoreDriver()

public DataStoreDriver()

The *DataStoreDriver* constructor is usually not called directly. Typically, JDBC drivers register themselves with the JDBC driver manager when their class is loaded (for example, when using *Class.forName*).

DataStoreDriver properties

Property	Implemented in
class*	java.lang.Object
majorVersion*	this class
minorVersion*	this class

majorVersion

public int getMajorVersion()

The part of the version number that indicates significant changes in the driver. For example, for version 1.02, the major version number is 1.

minorVersion

public int getMinorVersion()

The part of the version number that indicates maintenance changes in the driver. For example, for version 1.02, the minor version number is 2.

DataStoreDriver methods

Method	Implemented in
acceptsURL(java.lang.String)	this class
clone()	java.lang.Object
connect(java.lang.String, java.util.Properties)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getPropertyInfo(java.lang.String, java.util.Properties)	this class
hashCode()	java.lang.Object
jdbcCompliant()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object

Method	Implemented in
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

DataStoreServer component

datastore.jdbc package

Extends java.lang.Thread

Implements java.lang.Runnable

DataStoreServer is a thread that listens on a given port number for requests from the remote DataStore JDBC driver. Upon accepting a request, a connection is created that will service any further requests from that remote JDBC driver. This thread is the core of a DataStore JDBC server.

The server creates a *ServerConnection* object to represent each new connection. Objects that implement *ServerStatusListener* may register with the server to be notified when connections are made, connections are broken (disconnected), and when errors occur. You can control which types of events are reported.

Because access to a physical DataStore file is restricted to a single process, you must create your own custom DataStore server application if you want to add functionality. For example, in addition to servicing remote JDBC requests, you could have your server make backups at the same time every night. Another example would be if you want to access serialized file streams in the same DataStore file; these streams are not accessible through JDBC, but you could add that functionality through the DataStore API to your custom DataStore server.

DataStoreServer variables

Variable	Defined in
DEFAULT_PORT	this class
MAX_PRIORITY	java.lang.Thread
MIN_PRIORITY	java.lang.Thread
NORM_PRIORITY	java.lang.Thread

DEFAULT_PORT

public static final int DEFAULT_PORT = 2508

The default port number for a JDBC remote driver.

DataStoreServer constructors

DataStoreServer()

public DataStoreServer()

Creates a new *DataStoreServer* instance, which starts a thread that listens for activity on the default port.

DataStoreServer properties

Property	Implemented in
alive*	java.lang.Thread
class*	java.lang.Object
contextClassLoader	java.lang.Thread
daemon	java.lang.Thread
interrupted*	java.lang.Thread
name	java.lang.Thread
port	this class
priority	java.lang.Thread
reportConnect	this class
reportConnectError	this class
reportServerError	this class
serverConnections*	this class
tempDir	this class
threadGroup*	java.lang.Thread

port

public int getPort()
public void setPort(int port)

The port on which to listen for JDBC requests. If not set, *DataStoreServer.DEFAULT_PORT* is used.

reportConnect

```
public final boolean isReportConnect()
public final void setReportConnect(boolean set)
```

Determines whether connections and disconnections are reported to registered *ServerStatusListeners* or ignored.

reportConnectError

```
public final boolean isReportConnectError()
public final void setReportConnectError(boolean set)
```

Determines whether errors regarding the instantiation, connection, or disconnection of *ServerConnection* objects are reported to registered *ServerStatusListeners* or ignored.

reportServerError

```
public final boolean isReportServerError()
public final void setReportServerError(boolean set)
```

Determines whether (fatal) server errors are reported to registered *ServerStatusListeners* or ignored.

serverConnections

```
public final Vector getServerConnections()
```

Returns a list of current connections, represented by *ServerConnection* objects.

tempDir

```
public String getTempDir()
public void setTempDir(String tempDir)
```

The temporary directory for all *DataStore* connections. If **null**, the current directory is used.

DataStoreServer methods

Method	Implemented in
activeCount()	java.lang.Thread
addServerStatusListener(listener)	this class
checkAccess()	java.lang.Thread
clone()	java.lang.Object
closeConnections()	this class
countStackFrames()	java.lang.Thread

Method	Implemented in
currentThread()	java.lang.Thread
destroy()	java.lang.Thread
dumpStack()	java.lang.Thread
enumerate(java.lang.Thread[])	java.lang.Thread
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
interrupt()	java.lang.Thread
interrupted()	java.lang.Thread
join()	java.lang.Thread
join(long, int)	java.lang.Thread
join(long)	java.lang.Thread
notify()	java.lang.Object
notifyAll()	java.lang.Object
removeServerStatusListener(listener)	this class
resume()	java.lang.Thread
run()	this class
shutdown()	this class
sleep(long, int)	java.lang.Thread
sleep(long)	java.lang.Thread
start()	java.lang.Thread
stop()	java.lang.Thread
stop(java.lang.Throwable)	java.lang.Thread
suspend()	java.lang.Thread
toString()	java.lang.Thread
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
yield()	java.lang.Thread

addServerStatusListener(listener)

public final void addServerStatusListener(ServerStatusListener listener)

listener *ServerStatusListener* to be notified of status changes (like connections and disconnections) and exceptions encountered by the server.

closeConnections()

public final void closeConnections()

Close all *ServerConnection* instances initiated by this instance of *DataStoreServer*.

removeServerStatusListener(listener)

public final void removeServerStatusListener(ServerStatusListener listener)

listener *ServerStatusListener* to removed from the event notification list.

shutdown()

public final void shutdown()

Causes the *DataStoreServer* thread to terminate and close all *ServerConnection* instances initiated by this *DataStoreServer* instance.

ServerConnection class

datastore.jdbc package

Extends java.lang.Thread

Implements java.lang.Runnable

One of these threads is automatically instantiated and started by the *DataStoreServer* when it attempts to service a connection request, to represent that connection.

ServerConnection variables

Variable	Defined in
MAX_PRIORITY	java.lang.Thread
MIN_PRIORITY	java.lang.Thread
NORM_PRIORITY	java.lang.Thread

ServerConnection properties

Property	Implemented in
alive*	java.lang.Thread
class*	java.lang.Object
contextClassLoader	java.lang.Thread

Property	Implemented in
daemon	java.lang.Thread
interrupted*	java.lang.Thread
name	java.lang.Thread
priority	java.lang.Thread
threadGroup*	java.lang.Thread
userName*	this class

userName

public final String getUsername()

The name of the user associated with this connection.

See also *DataStoreConnection.userName* property

ServerConnection methods

Method	Implemented in
activeCount()	java.lang.Thread
checkAccess()	java.lang.Thread
clone()	java.lang.Object
close()	this class
countStackFrames()	java.lang.Thread
currentThread()	java.lang.Thread
destroy()	java.lang.Thread
dumpStack()	java.lang.Thread
enumerate(java.lang.Thread[])	java.lang.Thread
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
interrupt()	java.lang.Thread
interrupted()	java.lang.Thread
join()	java.lang.Thread
join(long, int)	java.lang.Thread
join(long)	java.lang.Thread
notify()	java.lang.Object
notifyAll()	java.lang.Object
resume()	java.lang.Thread
run()	this class
sleep(long, int)	java.lang.Thread
sleep(long)	java.lang.Thread

Method	Implemented in
start()	java.lang.Thread
stop()	java.lang.Thread
stop(java.lang.Throwable)	java.lang.Thread
suspend()	java.lang.Thread
toString()	java.lang.Thread
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
yield()	java.lang.Thread

close()

public final void close()

Closes this connection and terminates its associated thread.

ServerStatus interface

datastore.jdbc package

ServerStatus enumerates the *statusCode* property of a *ServerStatusEvent*, which reports connections and errors that occur with a *DataStoreServer*.

ServerStatus variables

Variable	Defined in
CONNECT	this class
CONNECT_ERROR	this class
DISCONNECT	this class
SERVER_ERROR	this class

CONNECT

static final int CONNECT = 0x4

A new connection was started successfully.

CONNECT_ERROR

static final int CONNECT_ERROR = 0x2

A connection failed.

DISCONNECT

static final int DISCONNECT = 0x5

A connection was broken.

SERVER_ERROR

static final int SERVER_ERROR = 0x1

An error in the server (not one of its connections) occurred.

ServerStatusEvent class

datastore.jdbc package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

ServerStatusEvent represents a status change or error in *DataStoreServer* or one of its connections. It is reported to all *ServerStatusListener* objects that register with the *DataStoreServer*.

ServerStatusEvent variables

Variable	Defined in
source	java.util.EventObject

ServerStatusEvent properties

Property	Implemented in
class*	java.lang.Object
error*	this class
exceptionChain*	com.borland.jb.util.DispatchableEvent
serverConnection*	this class
source*	java.util.EventObject
statusCode*	this class

error

public final Throwable getError()

If the event is due to an error, this method returns that error. If the event is not an error, this method returns **null**.

serverConnection

public final ServerConnection getServerConnection()

ServerConnection that is the source of this event, or **null** if the source is the *DataStoreServer*.

statusCode

public final int getStatusCode()

The type of event, one of the *ServerStatus* variables.

ServerStatusEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

Overrides com.borland.jb.util.DispatchableEvent.dispatch(java.util.EventListener)

ServerStatusListener interface

datastore.jdbc package

Extends java.util.EventListener

ServerStatusListener is an interface implemented by objects that want to be informed of status changes or errors in a *DataStoreServer* or one of its

connections. The objects must register through the *DataStoreServer.addServerStatusListener* method.

ServerStatusListener methods

Method	Implemented in
serverStatusMessage(status)	this class

serverStatusMessage(status)

public void serverStatusMessage(ServerStatusEvent status)

Called when a status change or error occurs.

dx.dataset package

The *dx.dataset* package contains classes and interfaces that provide basic data access. This package also defines base provider and resolver classes as well as an abstract *DataSet* class that is extended to other *DataSet* objects. These classes provide access to information stored in databases and other data sources.

This package includes functionality covering the three main phases of data handling:

Providing	General functionality to obtain data and manage local data sets. (JDBC specific connections to remote servers are handled by classes in the <i>com.borland.dx.sql.dataset</i> package.)
Manipulation	Navigation and editing of the data locally.
Resolving	General routines for the updating of data from the local <i>DataSet</i> back to the original source of the data. (Resolving data changes to remote servers through JDBC is handled by classes in the <i>com.borland.dx.sql.dataset</i> package.)

The *dx.dataset* package contains the following types of classes:

- Aggregate operator classes
- Column-related classes
- *DataSet* classes
- Descriptor classes
- Event, listener, and adapter classes
- Events for component writers
- Exception classes
- Import/export classes
- Provider and resolver classes
- Row-related classes
- Miscellaneous dataset classes

The following classes, components, and interfaces in this package are used internally by classes in this and other *com.borland* packages. These classes, components, and interfaces are not intended for general use and are not documented. Do not use them directly in your application.

- AggManager
- BoundsAggOperator
- Coercer
- ColumnDesigner
- DataIndex
- Designable
- DirectIndex
- Index
- IndexData
- InternalRow
- MasterNavigateEvent
- MasterNavigateListener
- MasterUpdateEvent
- MasterUpdateListener
- MatrixData
- RowVariant
- Store
- StoreInternals

The following classes in this package are BeanInfos:

- ColumnBeanInfo
- DataSetViewBeanInfo
- ParameterRowBeanInfo
- StorageDataSetBeanInfo
- TableDataSetBeanInfo
- TextDataFileBeanInfo

For more information, visit the database newsgroup. Details on newsgroups can be found at <http://www.borland.com/newsgroups>. The database newsgroup is dedicated to issues about writing database applications and is actively monitored by our support engineers as well as the Development team.

Interfaces

AccessListener	CalcAggFieldsListener	CalcFieldsListener
CalcType	CoerceFromListener	CoerceToListener
ColumnAware	ColumnChangeListener	ColumnDesigner
ColumnPaintListener	CustomPaintSite	DataChangeListener
DataModule	DataSetAware	Designable
DxDispatch	EditListener	ExceptionListener

LoadCancel	LoadListener	LoadRowListener
Locate	MasterNavigateListener	MasterUpdateListener
MetaDataUpdate	NavigationListener	OpenListener
ResolverListener	ResponseListener	RowFilterListener
RowStatus	Sort	StatusListener
Store	StoreClassFactory	StoreInternals
UpdateMode		

Classes and components

AccessEvent	AggDescriptor	AggManager
AggOperator	BoundsAggOperator	CalcAggFieldsAdapter
Coercer	Column	ColumnBeanInfo
ColumnChangeAdapter	ColumnList	ColumnPaintAdapter
ColumnVariant	CountAggOperator	CustomAggOperator
DataChangeAdapter	DataChangeEvent	DataFile
DataFileFormat	DataIndex	DataRow
DataSet	DataSetData	DataSetException
DataSetView	DataSetViewBeanInfo	DirectIndex
EditAdapter	ExceptionEvent	Index
IndexData	InternalRow	LoadEvent
MasterLinkDescriptor	MasterNavigateEvent	MasterUpdateEvent
MatrixData	MaxAggOperator	MinAggOperator
NavigationEvent	OpenAdapter	ParameterRow
ParameterRowBeanInfo	ParameterType	PickListDescriptor
Provider	ProviderHelp	ReadRow
ReadWriteRow	Resolver	ResolverAdapter
ResolverResponse	ResponseAdapter	ResponseEvent
RowFilterResponse	RowIterator	RowVariant
SortDescriptor	StatusEvent	StorageDataSet
StorageDataSetBeanInfo	SumAggOperator	TableDataSet
TableDataSetBeanInfo	TextDataFile	TextDataFileBeanInfo
ValidationException	Variant	VariantException

Overview of classes in the *com.borland.dx.dataset* package

Aggregate operator classes

AggOperator	Components that perform aggregate operations for aggregate columns. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .
CountAggOperator	Extends AggOperator. Used to maintain a count aggregation. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .
SumAggOperator	Extends AggOperator. Used to maintain a sum aggregation. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .
CustomAggOperator	Extends AggOperator. Used to maintain a custom aggregation via the implementation of the <i>CalcAggFieldsListener</i> methods. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .
MaxAggOperator	Extends AggOperator. Used to maintain a Max aggregation. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .
MinAggOperator	Extends AggOperator. Used to maintain a Min aggregation. Specified through the <i>aggDescriptor</i> property of a <i>Column</i> .

Column-related classes

Column	Stores column properties such as persistence, max and min values, alignment, and data type. Values stored in a <i>Column</i> are accessed through <i>DataSet</i> APIs.
ColumnVariant	Extends variant. Used in the context of data-aware controls. Provides the associated column and <i>DataSet</i> from which this value came.

DataSet classes

DataSet	An abstract class to provide a cursor for accessing and navigating table data. Manages a pseudo-record in memory to temporarily store a newly inserted row, or changes to the current row. Multiple data-aware controls can be bound to and synchronized with the same <i>DataSet</i> . Supports master-detail relationships. Supports model-view data-aware controls, to enable easy and flexible editing and navigation of data in a common way regardless of how the data was obtained.
StorageDataSet	Implements <i>DataSet</i> , to store data. Can obtain data from a remote server through the use of a query or stored procedure, or from a text file. After data is stored in a <i>StorageDataSet</i> , you can easily manipulate it and connect it to UI controls, without regard to which component is storing the data.
TableDataSet	Extends <i>StorageDataSet</i> . A simple <i>DataSet</i> with no formal provider or resolver of its data. Used to create a <i>StorageDataSet</i> from sources other than SQL databases.
CoerceFromListener	Used by implementors of <i>Providers</i> and <i>Resolvers</i> to coerce data from the type of the data source to the type of the <i>Column</i> . It is used by <i>QueryDataSet</i> , <i>ProcedureDataSet</i> , <i>QueryResolver</i> , and <i>ProcedureResolver</i> components.
CoerceToListener	Used by implementors of <i>Providers</i> and <i>Resolvers</i> to coerce data from the data type of the source to the data type of the <i>Column</i> . It is used by <i>QueryDataSet</i> , <i>ProcedureDataSet</i> , <i>QueryProvider</i> and <i>ProcedureProvider</i> components.
DataSetView	Presents alternate views and navigation of the data in the <i>DataSet</i> , without the need for multiple objects that each store data. Can display a different sort order and filtering than the source <i>StorageDataSet</i> .

Descriptor classes

SortDescriptor	Describes the order of presentation for rows of data that are visible to a <i>DataSet</i> . The <i>DataSet</i> can automatically reposition a new or updated row within the cursor, based on the ordering of data in specified columns.
AggDescriptor	Used to specify the grouping, target field to aggregate on, and the aggregation operation for a column with a <code>calcType</code> of <code>AGGREGATE</code> .
MasterLinkDescriptor	Stores properties that establish a master-detail relationship between two <i>DataSets</i> . You can link different <i>DataSets</i> together, such as a <i>QueryDataSet</i> and a <i>TableDataSet</i> , if there is common data to base the link relationship on.
PickListDescriptor	Used as a property setting for columns, to describe a picklist relationship between a column in one <i>DataSet</i> and separate picklist <i>DataSet</i> . When a row is selected, the values of the picklist columns in the picklist <i>DataSet</i> are copied into the destination columns.

Event, listener, and adapter classes

CalcFieldsListener	Used for performing calculations on row values.
CalcAggFieldsListener	Used for performing calculations on aggregated values.
CalcAggFieldsAdapter	Used for performing calculations on aggregated values.
ColumnChangeListener	For events related to editing column values.
ColumnChangeAdapter	For events related to editing column values.
ColumnPaintAdapter	For events related to the painting of a value in a <i>Column</i> at a specific row location.
ColumnPaintListener	The listener interface for events related to the painting of a value in a <i>Column</i> at a specific row location.
EditListener	Listens for events that are generated when a <i>DataSet</i> is edited, both before and after editing.
EditAdapter	An adapter class for <i>EditListener</i> .
ExceptionEvent	Used to override the <i>DataSet</i> error handling for data-aware controls.

ExceptionListener	A listener interface for <i>ExceptionEvent</i> .
LoadEvent	Indicates completion of a load operation on a <i>StorageDataSet</i> . Occurs when a query or procedure is executed, and when a <i>StorageDataSet</i> is loaded from an import operation. Of interest for queries, or procedures, that are executed with asynchronous fetching.
LoadListener	Listens for completion of a load operation on a <i>StorageDataSet</i> .
NavigationEvent	A notification that the <i>DataSet</i> 's cursor position has changed.
NavigationListener	A listener interface for <i>NavigationEvent</i> .
OpenListener	Notification that a <i>DataSet</i> is opening, has opened, is closing, or has closed.
OpenAdapter	An adapter class for <i>OpenListener</i> .
ResolverListener	Used to process insert-, delete-, and update-related events that occur during dataset resolution.
ResolverAdapter	An adapter class for <i>ResolverListener</i> .
ResponseEvent	Used to collect a response from the user.
ResponseListener	Listener class for <i>ResponseEvent</i> .
ResponseAdapter	An adapter class for <i>ResponseListener</i> .
RowFilterListener	Used to filter out rows for a <i>DataSet</i> 's view. To prevent rows from appearing in the storage of a <i>DataSet</i> , listen for an <i>EditListener.adding()</i> event.
StatusEvent	Used to inform listeners when specified types of status messages occur.
StatusListener	A listener interface for <i>StatusEvent</i> .

Events for component writers

AccessEvent	Internal event generated when a <i>DataSet</i> is opened, closed, or restructured.
AccessListener	Useful notifications for component writers. Not for general usage.
DataChangeEvent	The event object dispatched when data changes. Event types indicate the type of update.
DataChangeListener	Listener interface for the <i>DataChangeEvent</i> .
DataChangeAdapter	Adapter class for <i>DataChangeListener</i> .

Exception classes

<code>DataSetException</code>	A base Exception class used often throughout the <i>dataset</i> package.
<code>ValidationException</code>	Used often throughout the <i>dataset</i> package for column- and row-level validation errors that occur when posting changed rows or new rows of data. Extends <i>DataSetException</i> .
<code>VariantException</code>	An exception thrown when <i>Variant</i> runs into a problem.

Import/export classes

<code>DataFile</code>	This base class collects the behavior of all file-based data sources: importing data from a file and exporting data to a file. Extend this class for new classes defining a custom file format that you want to import data from, or export data to. Extended by <i>TextDataFile</i> .
<code>TextDataFile</code>	Used when importing data stored in a text format into a <i>TableDataSet</i> component, or when exporting the data stored in any <i>StorageDataSet</i> to a text file.
<code>DataFileFormat</code>	Contains localization variables to store whether data is stored as 8-bit ASCII characters, and whether conversions from locale-specific Unicode to multibyte character sets need to take place when reading and writing data. Often used by <i>TextDataFile</i> .

Provider and resolver classes

<code>DataSetData</code>	Provides the capability of using Java serialization to serialize the data in a <i>DataSet</i> .
<code>MetaDataUpdate</code>	Used to customize metadata discovery.
<code>Provider</code>	Abstract class that defines provider basics.
<code>ProviderHelp</code>	Collection of methods that are helpful to the providing or resolving phases.
<code>Resolver</code>	The <i>resolver</i> property setting for <i>DataSets</i> .
<code>ResolverResponse</code>	Used for collecting a response from another component.
<code>UpdateMode</code>	Used to specify the level of optimistic concurrency for row update operations.

Row-related classes

<code>DataRow</code>	Stores a single row of values across the columns of a <i>DataSet</i> . Useful for adding, updating and locating rows in a <i>DataSet</i> . Supports running a locate operation against specified columns, and handles columns of any data type in a <i>DataSet</i> .
<code>ParameterRow</code>	Useful for setting and getting <i>Query</i> and <i>StoredProcedure</i> parameter values. Allows a column to be included more than once in a data row. Useful for query parameters, to specify the same column multiple times, such as for multiple range comparisons.
<code>ParameterType</code>	Constants used by Column objects in <i>ParameterRows</i> .
<code>ReadRow</code>	Provides read-only row operations.
<code>ReadWriteRow</code>	Provides read and write row operations.
<code>RowFilterResponse</code>	Used for row filtering.
<code>RowStatus</code>	Stores status settings used by <i>DataSet</i> components, such as whether the row is inserted, updated or deleted.

Miscellaneous dataset classes

<code>CustomPaintSite</code>	Provides custom item painters and painters with information about the host container in which the painting occurs.
<code>CalcType</code>	Defines the types of calculations that a column may use: calculated values, aggregate values, lookups or no calculations.
<code>ColumnAware</code>	An interface that allows a component to declare to <i>JBuilder</i> that it knows how to bind to a specific column in a specific <i>DataSet</i> .
<code>DataModule</code>	An interface that you implement when creating a custom data module (data model). The data model is a container for non-visual components such as the <i>DataSet</i> and <i>Database</i> components, and contains business logic that controls how data is manipulated before and after the user or client sees the data.
<code>DataSetAware</code>	Permits a component to declare to <i>JBuilder</i> that it knows how to bind to a <i>com.borland</i> <i>DataSet</i> .

LoadCancel	Can be used to cancel a load operation. Especially useful for asynchronous queries. Can also be used to cancel a <i>TextDataFile</i> load operation.
Locate	Encapsulates commonly-used search options, such as search for the first, subsequent or last occurrence; case sensitivity; and closest match.
Variant	A type storage class used for the <i>dataset</i> package.

AccessEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

The *AccessEvent* is the internal event generated when a *DataSet* is opened, closed, or restructured. The *AccessEvent* class may be useful for component writers. Not for general usage.

The *AccessListener* class responds to the *AccessEvent* class.

AccessEvent variables

Variable	Defined in
CLOSE	this class
COLUMN_ADD	this class
COLUMN_CHANGE	this class
COLUMN_DROP	this class
COLUMN_MOVE	this class
DATA_CHANGE	this class
OPEN	this class
PROPERTY_CHANGE	this class
source	java.util.EventObject
STRUCTURE_CHANGE	this class
UNKNOWN	this class
UNSPECIFIED	this class

CLOSE

public static final int CLOSE = 2

Event type. Returned from *getID()*.

COLUMN_ADD

```
public static final int COLUMN_ADD = 3
```

Reason for an OPEN event. Structural change where a *Column* was added to the *DataSet*.

COLUMN_CHANGE

```
public static final int COLUMN_CHANGE = 5
```

Reason for an OPEN event. Structural change where a *Column* was changed in the *DataSet*.

COLUMN_DROP

```
public static final int COLUMN_DROP = 4
```

Reason for an OPEN event. Structural change where a *Column* was dropped from the *DataSet*.

COLUMN_MOVE

```
public static final int COLUMN_MOVE = 6
```

Reason for an OPEN event. Structural change where a *Column* was moved in the *DataSet*.

DATA_CHANGE

```
public static final int DATA_CHANGE = 2
```

Reason for an OPEN event. Indicates that data has changed, but no structural changes were made. This is used when a *DataSet*'s *sort* property is changed or a *DataSet.empty()* is called. It is also used when *DataSet.enableDataSetEvents(true)* is called.

OPEN

```
public static final int OPEN = 1
```

Event type. Returned from *getID()*.

PROPERTY_CHANGE

```
public static final int PROPERTY_CHANGE = 9
```

Will get pertinent information when a following OPEN event occurs. Reason for CLOSE event. This is called when a non-structural property change like Column Font or Color is changed. Also used when *DataSet.enableDataSetEvents(false)* is called. Note that this event will not be sent to a *DataSetView* that is listening to its associated *StorageDataSet*. In practice, this event only makes its way to visual components that listen to Access events.

STRUCTURE_CHANGE

```
public static final int STRUCTURE_CHANGE = 8
```

Reason for CLOSE event. Indicates that *DataSet* was closed and it is expected to be reopened in a short time period. Usually called to perform a sort order change or structural property change.

UNKNOWN

```
public static final int UNKNOWN = 7
```

Reason for a CLOSE event. Indicates that *DataSet* was closed and it is not known whether it will be opened again. Calling *DataSet.close()* will cause this event.

UNSPECIFIED

```
public static final int UNSPECIFIED = 1
```

Reason for an OPEN event. Unknown *DataSet.open()* will cause this. Returned from *getReason()*.

AccessEvent constructors

AccessEvent(java.lang.Object, com.borland.dx.dataset.AccessEvent)

```
public AccessEvent(Object source, AccessEvent event)
```

Creates an internal event from the given source for the specified event.

source The event source.

event The type of event that changed a *DataSet*.

AccessEvent(java.lang.Object, int)

```
public AccessEvent(Object source, int id)
```

Creates an internal event from the given source with the specified event type.

source The event source.

id The event type: An OPEN event is 1; a CLOSE event is 2.

AccessEvent(java.lang.Object, int, int)

public AccessEvent(Object source, int id, int reason)

Creates an internal event from the given source with the specified event type and reason.

<i>source</i>	The event source.
<i>id</i>	The event type: An OPEN event is 1; a CLOSE event is 2.
<i>reason</i>	The reason for the event. Reasons for OPEN events include: <ul style="list-style-type: none"> • UNSPECIFIED = 1 • DATA_CHANGE = 2 • COLUMN_ADD = 3 • COLUMN_DROP = 4 • COLUMN_CHANGE = 5 • COLUMN_MOVE = 6 Reasons for CLOSE events include: <ul style="list-style-type: none"> • UNKNOWN = 7 • STRUCTURE_CHANGE = 8 • PROPERTY_CHANGE = 9

AccessEvent(java.lang.Object, int, int, com.borland.dx.dataset.Column)

public AccessEvent(Object source, int id, int reason, Column dropColumn)

Creates an internal event from the given source with the specified event type and reason. The internal event drops the specified column.

<i>source</i>	The event source.
<i>id</i>	The event type: An OPEN event is 1; a CLOSE event is 2.
<i>reason</i>	The reason for the event. Reasons for OPEN events include: <ul style="list-style-type: none"> • UNSPECIFIED = 1 • DATA_CHANGE = 2 • COLUMN_ADD = 3 • COLUMN_DROP = 4 • COLUMN_CHANGE = 5 • COLUMN_MOVE = 6 Reasons for CLOSE events include: <ul style="list-style-type: none"> • UNKNOWN = 7 • STRUCTURE_CHANGE = 8 • PROPERTY_CHANGE = 9
<i>dropColumn</i>	The name of the column being dropped.

AccessEvent(java.lang.Object, int, int, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)

public AccessEvent(Object source, int id, int reason, Column oldColumn, Column newColumn)

Creates an internal event from the given source with the specified event type and reason. The internal event restructures the table.

<i>source</i>	The event source.
<i>id</i>	The event type: An OPEN event is 1; a CLOSE event is 2.
<i>reason</i>	The reason for the event. Reasons for OPEN events include: <ul style="list-style-type: none"> • UNSPECIFIED = 1 • DATA_CHANGE = 2 • COLUMN_ADD = 3 • COLUMN_DROP = 4 • COLUMN_CHANGE = 5 • COLUMN_MOVE = 6 Reasons for CLOSE events include: <ul style="list-style-type: none"> • UNKNOWN = 7 • STRUCTURE_CHANGE = 8 • PROPERTY_CHANGE = 9
<i>oldColumn</i>	The name of the column to replace.
<i>newColumn</i>	The name of the column to replace <i>oldColumn</i> with.

AccessEvent(java.lang.Object, int, int, int, int)

public AccessEvent(Object source, int id, int reason, int oldOrdinal, int newOrdinal)

Creates an internal event from the given source with the specified event type and reason. The internal event restructures the table.

<i>source</i>	The event source.
<i>id</i>	The event type: An OPEN event is 1; a CLOSE event is 2.
<i>reason</i>	The reason for the event. Reasons for OPEN events include: <ul style="list-style-type: none"> • UNSPECIFIED = 1 • DATA_CHANGE = 2 • COLUMN_ADD = 3 • COLUMN_DROP = 4 • COLUMN_CHANGE = 5 • COLUMN_MOVE = 6 Reasons for CLOSE events include: <ul style="list-style-type: none"> • UNKNOWN = 7 • STRUCTURE_CHANGE = 8 • PROPERTY_CHANGE = 9

oldOrdinal The ordinal number of the column to replace.

newOrdinal The ordinal number of the column to replace *oldOrdinal* with.

AccessEvent properties

Property	Implemented in
class*	java.lang.Object
dropColumn*	this class
exceptionChain*	com.borland.jb.util.DispatchableEvent
ID*	this class
newColumn*	this class
newOrdinal*	this class
oldColumn*	this class
oldOrdinal*	this class
reason*	this class
source*	java.util.EventObject

dropColumn

```
public Column getDropColumn()
```

Read-only property that returns information about which *Column* was dropped from a *COLUMN_DROP* operation.

ID

```
public final int getID()
```

The event type. Valid return values for this method are defined as variables in this class.

newColumn

```
public Column getNewColumn()
```

Read-only property that returns information on a new *Column* from a *COLUMN_CHANGE* operation.

newOrdinal

```
public int getNewOrdinal()
```

Read-only property that returns the new ordinal position when a *Column* is moved.

oldColumn

```
public Column getOldColumn()
```

Read-only property that returns the old *Column* from a *COLUMN_CHANGE* operation.

oldOrdinal

```
public int getOldOrdinal()
```

Read-only property that returns the previous ordinal position when a *Column* is moved.

reason

```
public final int getReason()
```

Read-only property that returns the reason for the event. Valid values are defined in the variables section of this class.

AccessEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

```
public void dispatch(EventListener listener)
```

This method is used internally by other *com.borland* classes. You should never use this method directly.

Overrides com.borland.jb.util.DispatchableEvent.dispatch(java.util.EventListener)

AccessListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.DataSet, com.borland.dx.dataset.DataSetView, com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.TableDataSet, com.borland.dx.sql.dataset.ProcedureDataSet, com.borland.dx.sql.dataset.QueryDataSet

The *AccessListener* interface provides notification when a *DataSet* is opened, closed, or restructured. Useful for component writers. Not for general usage.

AccessListener methods

Method	Implemented in
accessChange(com.borland.dx.dataset.AccessEvent)	this class

accessChange(com.borland.dx.dataset.AccessEvent)

public void accessChange(AccessEvent event)

Provides information regarding how a *DataSet* has been changed.

event The type of event that changed a data set: opened, closed, or restructured. See *com.borland.dx.dataset.AccessEvent* for more information.

AggDescriptor class

dx.dataset package

Extends java.lang.Object

Implements java.io.Serializable

The *AggDescriptor* class collects the properties associated with an aggregation calculation. It is associated with a *Column* by the *Column* component's *agg* property. In the JBuilder Inspector, this object is accessed from the *agg* property. Selecting this property displays the Agg property editor where you can define the properties for your aggregate operation. This editor allows you to define:

- the column or columns specifying the group of rows on which to aggregate
- the column containing the value to aggregate
- the aggregate operation (Sum, Min, Max or Count)
- optionally specify that a custom aggregation is to be performed

Set the *agg* property of a *Column* component to an instance of this class to define the type of aggregation to perform: count, sum, minimum value or maximum value. You specify a single column for the aggregation operation. You can optionally include a set of *Column* names to base the grouping logic of the aggregation (subtotals) or, if left unspecified, the calculation is maintained across an entire *DataSet*.

The update order of calc, lookup and aggregate columns is as follows:

- 1 All lookup values are initialized in the row that is passed to the CalcFields event handler. Calculations on aggregates must be performed in the *calcAggAdd()* and *calcAggDelete()* event handlers.
- 2 The *calcAggAdd* and *calcAggDelete* event handlers are both called after the non-calculated aggregates have been updated for add or delete operations. An update operation is performed as a delete and then followed by an add operation.

Note

DataSet.addRow() and *DataSet.updateRow()* methods will be modified with the appropriate lookup values for any lookup columns they may contain before the calcFields event is called.

AggDescriptor constructors

AggDescriptor(java.lang.String[], java.lang.String, com.borland.dx.dataset.Aggregator)

```
public AggDescriptor(String[] groupColumnNames, String aggColumnName, Aggregator aggregator)
```

Constructs an *AggDescriptor* object with the specified parameters.

- | | |
|-------------------------|---|
| <i>groupColumnNames</i> | An array of <i>Column</i> names that define the groups of rows to aggregate on. |
| <i>aggColumnName</i> | The name of the <i>Column</i> to perform the aggregation on. |
| <i>aggOperator</i> | The aggregation operation to perform. See the <i>aggOperator</i> property for valid values. |

AggDescriptor properties

Property	Implemented in
aggColumnName*	this class
aggOperator*	this class
class*	java.lang.Object
groupColumnNames*	this class

aggColumnName

public String getAggColumnName()

Read-only property that returns the *Column* (by name) to perform the aggregation on.

aggOperator

public AggOperator getAggOperator()

Read-only property that specifies the aggregation operation to perform. Valid values are:

- *SumAggOperator*
- *MinAggOperator*
- *MaxAggOperator*
- *CountAggOperator*
- Any custom aggregation defined by extending the *CustomAggOperator* class

groupColumnNames

public String[] getGroupColumnNames()

Read-only property that returns the array of *Column* names that define the groups of rows to aggregate on.

AggDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

AggOperator class (abstract)

dx.dataset package

Extends java.lang.Object

Extended by com.borland.dx.dataset.BoundsAggOperator,
com.borland.dx.dataset.CountAggOperator,
com.borland.dx.dataset.CustomAggOperator,
com.borland.dx.dataset.SumAggOperator

Implements java.io.Serializable, java.lang.Cloneable

The *AggOperator* class is an abstract class that defines basic aggregator behavior of a calculated value or a calculated *Column*. All aggregation operators must extend from this class, therefore, you should extend this class when creating custom *AggOperator* classes. The following aggregator operators are provided and can be specified as the *aggOperator* of the *AggDescriptor* object:

- *SumAggOperator*
- *CountAggOperator*
- *MinAggOperator* (a subclass of *BoundsAggOperator*)
- *MaxAggOperator* (a subclass of *BoundsAggOperator*)

AggOperator variables

Variable	Defined in
aggColumn	this class
aggDataSet	this class
aggValue	this class
dataSet	this class
resultColumn	this class
resultValue	this class

aggColumn

protected transient Column aggColumn

The *Column* in the *DataSet* that is being aggregated on.

aggDataSet

protected transient DataSet aggDataSet

The internal *StorageDataSet* that contains and maintains aggregated values.

aggValue

protected transient Variant aggValue

Pre-allocated storage for an *aggColumn* value. *AggOperator* extension classes use this for storing and manipulating an *aggColumn* value.

dataSet

protected transient StorageDataSet dataSet

The *StorageDataSet* that contains the *Column* (“aggColumn”) that is being aggregated on.

resultColumn

protected transient Column resultColumn

The *Column* in *aggDataSet* that contains the aggregated value.

resultValue

protected transient Variant resultValue

Pre-allocated storage for a *resultColumn* value. *AggOperator* extension classes use this for storing and manipulating a *resultColumn* value.

AggOperator properties

Property	Implemented in
class*	java.lang.Object

AggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	this class
clone()	this class
delete(com.borland.dx.dataset.ReadRow, long)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	this class
locate(com.borland.dx.dataset.ReadRow)	this class
needsAggDataSet()	this class

Method	Implemented in
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add(com.borland.dx.dataset.ReadRow, long, boolean)

public abstract void add(ReadRow row, long internalRow, boolean first)

A row has been added or updated.

- row* The row containing the values.
- internalRow* The unique identifier for the row.
- first* Returns **true** if this is the first row in the group, **false** otherwise.

clone()

public Object clone()

Returns a clone (or copy) of this *AggOperator*.

Overrides java.lang.Object.clone()

delete(com.borland.dx.dataset.ReadRow, long)

public abstract void delete(ReadRow row, long internalRow)

A row has been deleted or updated. The *row* parameter contains the row's values and *internalRow* is a unique identifier for that row.

init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)

public void init(StorageDataSet dataSet, String[] groupColumnNames, StorageDataSet aggDataSet, Column resultColumn, Column aggColumn)

Called when the *AggOperator* is being initialized. If overridden by an extended class, *super.init()* should still be called to ensure proper initialization. Note that this method is called before the *aggDataSet* is opened. This allows an *AggOperator* extension to add a column to the *aggDataSet*. This

can be useful for some types of maintained aggregations like average that would need to accumulate count and total.

<i>dataSet</i>	The <i>StorageDataSet</i> that contains the <i>aggColumn</i> that is being aggregated on
<i>groupColumnNames</i>	A array of the <i>Column</i> names to perform grouping by.
<i>aggDataSet</i>	The internal <i>StorageDataSet</i> that contains and maintains aggregated values.
<i>resultColumn</i>	The <i>Column</i> in <i>aggDataSet</i> that contains the aggregated value.
<i>aggColumn</i>	The <i>Column</i> in the <i>DataSet</i> that is being aggregated on.

locate(com.borland.dx.dataset.ReadRow)

```
public boolean locate(ReadRow row)
```

Implemented by subclasses *MinAggOperator* and *MaxAggOperator* to locate the minimum or maximum value for the grouping column values specified in *row*.

needsAggDataSet()

```
public boolean needsAggDataSet()
```

This method returns **true** and specifies whether the *AggOperator* subclass requires an *aggDataSet* (an internal *StorageDataSet* that contains and maintains aggregated values). The *MinAggOperator* and *MaxAggOperator* classes internally override this setting since they use a secondary index to track minimum and maximum values rather than an *aggDataSet*.

open(com.borland.dx.dataset.DataSet)

```
public void open(DataSet aggDataSet)
```

Called when the *aggDataSet* is opened and prepared for usage.

CalcAggFieldsAdapter class

dx.dataset package

Extends	java.lang.Object
Implements	com.borland.dx.dataset.CalcAggFieldsListener, java.util.EventListener
	This class is an adapter class for <i>CalcAggFieldsListener</i> . It is used for performing calculations on aggregated columns.

CalcAggFieldsAdapter properties

Property	Implemented in
class*	java.lang.Object

CalcAggFieldsAdapter methods

Method	Implemented in
calcAggAdd(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow)	this class
calcAggDelete(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow)	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

CalcAggFieldsListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.CalcAggFieldsAdapter

This interface is used for notification of a row being added or deleted on a *DataSet* having a calculation on an aggregate column. The *CalcAggFieldsListener* listens for this notification in order to recompute the calculation based on the aggregation.

For example, your application uses a sum aggregator to compute the total value of an order as the sum of the extended price column for all the rows for that order. It then uses a *CalcAggFieldsListener* to compute the tax and shipping charges on top of the total. When a row is added or deleted from

the order, the *CalcAggFieldsListener* is aware of this and recalculates the tax and shipping charges. In this example, you

- 1 Set a column's *calcType* to *CalcType.Aggregate*.
- 2 Supply an *aggDescriptor* to specify sorting (not target column or aggregation operation).
- 3 Add a *CalcAggFieldsListener* to compute the calculation based on the aggregated column.

This class differs from the *CalcFieldsListener* in that this class is used to do calculations on maintained aggregates. The *CalcFieldsListener* class is used for calculations on rows.

Once this listener is set on a *StorageDataSet*, the name of the listener class is remembered. If another *StorageDataSet* opens the same table stream without having set a listener with the same class name, the *StorageDataSet* will be marked *readOnly*. The *StorageDataSet* can be made writable again by doing one of the following:

- Add a listener with the correct class name.
- Call *StorageDataSet.recalc()* or *StorageDataSet.restructure()*. This will cause the system to forget the listener class name.

For more information, see *DataSetException.NEEDS_RECALC*.

CalcAggFieldsListener methods

Method	Implemented in
<code>calcAggAdd(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow)</code>	this class
<code>calcAggDelete(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow)</code>	this class

calcAggAdd(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow)

`public void calcAggAdd(ReadRow row, ReadWriteRow resultRow)`

Called when a new or modified row is posted. In the case of a modified row, both *calcAggDelete()* and *calcAggAdd()* are called.

row

The contents of the newly added or modified row.

resultRow

The values you want posted to the row. When *calcAggAdd()* is called, *resultRow* has the same values as *row*. Typically, you only change the values in columns that are calculations on aggregate columns.

**calcAggDelete(com.borland.dx.dataset.ReadRow,
com.borland.dx.dataset.ReadWriteRow)**

public void calcAggDelete(ReadRow row, ReadWriteRow resultRow)

Called when a row is deleted from the data set or a modified row is posted. In the case of a modified row, *calcAggAdd()* is also called.

<i>row</i>	The contents of the deleted or modified row.
<i>resultRow</i>	The contents of the deleted or modified row, including any changes you make to the columns that are calculations on aggregate columns. Values that you place in these columns will be duplicated into other rows in the same group before the row is deleted.

CalcFieldsListener interface

dx.dataset package

Extends java.util.EventListener

This interface is used for notification that a row that contains a calculated column has been added to a *DataSet*, or that a row that contains a calculated column in a *DataSet* has been modified. When the notification is received, the listener can initiate a recalculation of that row.

This class differs from the *CalcAggFieldsListener* in that this class is used to do calculations on rows. The *CalcAggFieldsListener* class is used for calculations on aggregates.

Once this listener is set on a *StorageDataSet*, the name of the listener class is remembered. If another *StorageDataSet* opens the same table stream without having set a listener with the same class name, the *StorageDataSet* will be marked *readOnly*. The *StorageDataSet* can be made writable again by doing one of the following:

- Add a listener with the correct class name.
- Call *StorageDataSet.recalc()* or *StorageDataSet.restructure()*. This will cause the system to forget the listener class name.

For more information, see *DataSetException.NEEDS_RECALC*.

CalcFieldsListener methods

Method	Implemented in
calcFields(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, boolean)	this class

calcFields(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, boolean)

public void calcFields(ReadRow changedRow, DataRow calcRow, boolean isPosted)

Called whenever a field value is modified or added in the listener's data set. The *isPosted* parameter indicates that the row is being posted. You may not want to recalculate fields on rows that are not yet posted.

<i>changedRow</i>	The current values in the row, including the field that was just added or modified.
<i>calcRow</i>	The values to be left in the row when <i>calcFields</i> returns. Typically, the code in <i>calcFields</i> only modifies values in the calculated columns of <i>calcRow</i> .
<i>isPosted</i>	Indicates whether or not the row is being posted.

CalcType interface

dx.dataset package

The *CalcType* interface defines the types of calculations that a *Column* may involve: no calculations, a calculated value, an aggregate value, or a lookup. Use the constants defined in this class with the *calcType* property of a *Column* component.

When you set the *calcType* property to *AGGREGATE*, you must also set the associated aggregation properties in the *AggDescriptor* object. These properties indicate the *Column* to perform the aggregation on, subtotal grouping, and the type of aggregator operation (sum, count, minimum value or maximum value). You access the *AggDescriptor* object through the *agg* property in the JBuilder Inspector.

When working with calculated fields, set the *Column* component's *calcType* property to *CALC*, then set the code for the calculation in the *Column* component's *calcFields* event handler. The *calcFields* event handler method is called for calculated columns whenever a field value is set and whenever a row is posted.

When working with lookup fields, set the *Column* component's *calcType* property to *LOOKUP* and its *pickList* property to a *PickListDescriptor* which describes the lookup relationship. In particular, the *PickListDescriptor* property of *LookupDisplayColumn* must be set to the *Column* to display the lookup values.

CalcType variables

Variable	Defined in
AGGREGATE	this class
CALC	this class
LOOKUP	this class
NO_CALC	this class

AGGREGATE

public static final int AGGREGATE = 2

Constant used to designate a calculated field that summarizes across multiple rows. To work with an aggregation calculation, set the *agg* property of the *Column* to the *AggDescriptor* object that contains the properties associated with the aggregation.

CALC

public static final int CALC = 1

Constant used to designate a basic calculated field that is updated by the *calcFields* event of a *Column* when rows are changed or added.

LOOKUP

public static final int LOOKUP = 3

Constant used to designate that this column gets its value from a *Column* in another *DataSet*. The *Column.PickList* property must be set with the *PickList.LookupDisplayColumn* set to a non-null value for the lookup to work.

NO_CALC

public static final int NO_CALC = 0

Constant used to designate that a calculation is not used for this *Column*.

CoerceFromListener interface

dx.dataset package

Extends

java.util.EventListener

If set, the *CoerceFromListener* interface is called by implementors of *Resolvers* to coerce data from the data type of the *Column* to the data type of the data source. It is used by *QueryDataSet*, *ProcedureDataSet*, *QueryResolver*, and

ProcedureResolver components. This allows an application to coerce column values from one data type to another as the values are received from a data source such as a JDBC driver.

Data types are automatically converted during the resolving phase according to the table listed in Data type conversions during data resolving. If you set a column's *dataType* property, thereby over-riding the default JDBC-to-*com.borland* mapping, automatic data type coercion is done. The exception to the automatic type coercion is when converting to and from a String to any type other than String (and other conversions involving such differing data types). A *VariantException* is thrown in these cases.

To customize the coercion, or to prevent the *VariantException* from being thrown (in cases as described in the previous paragraph), wire the *CoerceFromListener.coerceFromColumn(...)* event.

See also *Data type conversion during data providing*,
CoerceToListener.coerceToColumn(...) event

CoerceFromListener methods

Method	Implemented in
<code>coerceFromColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)</code>	this class

`coerceFromColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)`

`void coerceFromColumn(StorageDataSet dataSet, Column column, Variant from, Variant to)`

Allows an application to control the coercion of a data value from a column in the *DataSet* to the data source.

<i>dataSet</i>	The <i>DataSet</i> in which the column exists.
<i>Column</i>	The name of the column in <i>dataSet</i> .
<i>from</i>	The data value from the data source which needs to be coerced to the <i>to</i> parameter.
<i>to</i>	The column value that the <i>from</i> parameter must be coerced to.

CoerceToListener interface

dx.dataset package

Extends java.util.EventListener

If set, the *CoerceToListener* interface is called by implementors of *Providers* to coerce data from the data type of the data source to the data type of the *Column* component. It is used by *QueryDataSet*, *ProcedureDataSet*, *QueryProvider* and *ProcedureProvider* components. This allows an application to coerce column values from one data type to another as the values are sent to a data source, such as a JDBC driver.

Data types are automatically converted during the providing phase according to the table listed in Data type conversions during data providing. If you set a column's *dataType* property, thereby over-riding the default JDBC-to-*com.borland* mapping, automatic data type coercion is done. The exception to the automatic type coercion is when converting to and from a String to any data type other than String (and other conversions involving such differing data types). A *VariantException* is thrown in these cases.

To customize the coercion, or to prevent the *VariantException* from being thrown in cases as noted in the previous paragraph, wire the *CoerceToListener.coerceToColumn(...)* event.

See also Data type conversion during resolving,
CoerceFromListener.coerceFromColumn(...) event

CoerceToListener methods

Method	Implemented in
<code>coerceToColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)</code>	this class

`coerceToColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)`

`void coerceToColumn(StorageDataSet dataSet, Column column, Variant from, Variant to)`

Allows an application to control the coercion of a data value from the data source to a column value in *dataSet*.

- dataSet* The *DataSet* in which the *Column* exists.
- Column* The name of the column in *dataSet* parameter.

- from* The data value from the data source which needs to be coerced to the *to* parameter.
- to* The column value that the *from* parameter must be coerced to.

Column component

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.Designable, java.io.Serializable, java.lang.Cloneable

The *Column* component stores important column-level metadata type properties (such as data type and precision) as well as visual properties such as font and alignment. For *QueryDataSet* and *ProcedureDataSet* components, a set of *Column* components is dynamically created each time the *StorageDataSet* is instantiated, mirroring the actual columns in the data source at that time.

Data-aware controls pick up *Column* properties when bound to a *DataSet*. After that, to affect display, set properties of the *ColumnView* or *DataSetColumnView* components.

Calculated columns

A *Column* may derive its values as a result of a calculated expression. Values for the *Column* are calculated for each row of data using a formula defined in the *StorageDataSet* object's *calcFields* event handler. Calculated columns are read-only; you cannot set individual values of a row for columns that are calculated columns.

Typically, the formula for a calculated field uses expressions involving values from other *Columns* in that row to generate a value for each row of the calculated *Column*. For example, a table might have non-calculated columns for Quantity and UnitPrice, and a calculated *Column* for ExtendedPrice, which is calculated by multiplying the values of Quantity and UnitPrice. Calculated *Columns* are also useful for performing lookups in other tables. For example, a part number can be used to retrieve a part description for display in an invoice line item.

DataExpress calculated columns are read-only, and while JDataStore automatically recognizes this and attempts to discover other non-updateable columns, some JDBC drivers seem unaware that server-side calculated columns are read-only. For this reason, you should set column properties to read-only and not resolvable. While this seems redundant, this can prove crucial to your application.

Column persistence and order

An important *Column* property is persistence. Set this property to create a persistent, unchanging set of columns. Setting a *Column* to persistent guarantees that each time your application runs, it uses and displays the same *Column* components every time and in the same order.

Programmatically, you set the *persist* property.

You can define events at the *Column* level, however, you must make a column persistent before you can define events on it.

A side-effect of setting a *Column* as persistent is that persistent columns may not be placed in the same order as they are found in the *DataSet*.

When a *StorageDataSet* is provided data, it:

- Moves persistent columns to the left, as the first columns of the *DataSet*. Any non-persistent columns are deleted.
- Merges columns from the provided data with persistent columns. If a persistent column has the same name as a provided column it is considered to be the same column. If the data types do not match, the values will be coerced to a valid data type.
- Adds the remaining columns that are defined only in the application to the *DataSet*, in the order they are defined in the *DataSet.setColumns()* method call.
- Adds the provided columns in the order specified in the query or procedure.
- Attempts to place every column whose *preferredOrdinal* property is set to its desired location.

Column constraints

Another important *Column* level feature is the support of constraints. Supported constraints include:

- precision and scale
- minimum and maximum values
- default values

These constraints are not enforced when data is loaded into a *StorageDataSet* component since it is assumed that the data satisfied the constraints before being stored in the SQL database. However, you cannot leave a modified field until its value satisfies the constraints set on that *Column*. This is done automatically; you do not need special code to handle or enforce these constraints. You can extend constraints by writing a class that implements *ColumnChangeListener* or *EditListener* if you want additional validation handling.

Java Object support

Columns can contain various data types of data including *Object*. In general, you should provide custom *ItemPainter* and *ItemEditor* classes for each Java Object type. Set each Column that contains Object data to its applicable *ItemPainter* and *ItemEditor* classes.

The default *ItemPainter* prints the *toString()* value of the object. The default *ItemEditor* treats the Object as a String. The result of an edit will therefore be an Object of `java.lang.String`. If these defaults are acceptable given your Object's data, with the exception that the edited Object should be of a different data type, set a custom *formatter* for this Column. This formatter should be able to format the Object into a String and parse a String back into the wanted Object type.

Data type conflicts between a column and its data source

Some columns in a *DataSet* are provided, for example, by execution of a query or stored procedure. A provided column's *dataType* property is set by mapping the JDBC data type (stored in the *sqlType* property of the Column component) to the equivalent *Variant* data type (stored in the Column's *dataType* property).

You can override the default data type mapping by setting the Column's *dataType* property. By default, the *QueryProvider* and *QueryResolver* used by a *QueryDataSet* performs automatic type coercions between the various time and numeric data types as data is retrieved from and saved to a JDBC-based data source. However, there are some type coercions which are not automatically handled. This includes coercions to or from *String*, *Object*, and *InputStream* data types. In addition, the automatic coercions may not be acceptable for your application. For example, the coercion from a double to *BigDecimal* may incur undesirable precision loss. An application can override the automatic type coercion by wiring the *CoerceToListener.coerceToColumn(...)* and *CoerceToListener.coerceFromColumn(...)* events. This feature allows for greater portability of an application. For example, code such as

```
customerDS.setDate("HIRE_DATE",...);
```

could run on any server, even one that supports `TIMESTAMP` but not `DATE`.

Warning Be careful when assigning data types to your columns. Coercions, whether intentional or not, can cause data loss. For example, coercion from a `FLOAT` data source to an `INT` column type discards the fractional part of the value. If you save the value back to the server, the `INT` is converted back to a `FLOAT` thereby losing the fractional part of the data.

Column constructors

Column()

public Column()

Constructs a *Column* component with the following defaults:

Property	Value
<i>visible</i>	TriStateProperty.DEFAULT
<i>dataType</i>	Variant.ASSIGNED_NULL
<i>searchable</i>	true
<i>resolvable</i>	TriStateProperty.DEFAULT
<i>editable</i>	true

Column(java.lang.String, java.lang.String, int)

public Column(String columnName, String caption, int dataType)

Constructs a *Column* component with property values similar to those of its null constructor (*Column()*) and as specified by its parameters.

<i>columnName</i>	The <i>String</i> name of the <i>Column</i> . This name must be unique across all <i>Column</i> components in the <i>DataSet</i> .
<i>caption</i>	The <i>String</i> label used when displaying the <i>Column</i> in a data-aware control. The default caption is the <i>columnName</i> .
<i>dataType</i>	The data type of the items the <i>Column</i> will contain. Accepted values are defined in <i>com.borland.dx.dataset.Variant</i> variables

Column properties

Property	Implemented in
agg	this class
alignment	this class
background	this class
calcType	this class
caption	this class
class*	java.lang.Object
columnChangeListener*	this class
columnName	this class
columnPaintListener*	this class

Property	Implemented in
currency	this class
dataSet*	this class
dataType	this class
default	this class
defaultValue	this class
displayMask	this class
editable	this class
editMask	this class
editMasker	this class
exportDisplayMask	this class
exportFormatter	this class
fixedPrecision	this class
font	this class
foreground	this class
formatter	this class
hidden	this class
itemEditor	this class
itemPainter	this class
javaClass	this class
locale	this class
max	this class
maxInline	this class
maxValue	this class
min	this class
minValue	this class
ordinal*	this class
parameterType	this class
persist	this class
pickList	this class
precision	this class
preferredOrdinal	this class
readOnly	this class
required	this class
resolvable	this class
rowId	this class
scale	this class
schemaName	this class
searchable	this class
serverColumnName	this class
sortable*	this class
sortPrecision	this class

Property	Implemented in
sqlType	this class
tableName	this class
textual*	this class
visible	this class
width	this class

agg

```
public final AggDescriptor getAgg()
public final void setAgg(AggDescriptor aggDescriptor)
```

Stores the *AggDescriptor* object that defines the aggregator properties for a calculated *Column* whose *calcType* property is *AGGREGATE*.

alignment

```
public final int getAlignment()
public final void setAlignment(int alignment)
```

Stores the alignment of the items in the *Column*, both horizontal and vertical. To set a new value for both horizontal and vertical alignment, separate the values with a vertical bar (|). Unless otherwise specified, *alignment* defaults to center and bottom. Acceptable values for *alignment* are defined in *com.borland.dx.text.Alignment* variables.

background

```
public final Color getBackground()
public final void setBackground(Color background)
```

Stores the background color used by the *Column* component in UI controls. Any color object is valid. The *java.awt.color* description provides class variables for some common colors.

calcType

```
public final int getCalcType()
public final void setCalcType(int calcType)
```

Specifies the calculation type of the *Column*. Valid values for the calculation type are defined in *com.borland.dx.dataset.CalcType* variables. Setting a column to any calculation type (other than *NO_CALC*) sets this *Column* to *readOnly*. On error, the *setCalcType()* method throws a *DataSetException*.

The *DataSet* must be closed in order to set this property.

caption

```
public final String getCaption()
public final void setCaption(String caption)
```

Stores the *Column* object's label that displays in a data-aware control. The value of the caption need not be unique across all *Column* objects of the *DataSet*. The default caption is the *Column* name.

columnChangeListener

```
public final ColumnChangeListener getColumnChangeListener()
```

This is the listener to use for checks on field values that should be done before a user leaves the field. One example of when this might be useful is to check that a part number is in stock before the rest of the line item information is entered. Writing a listener for a *StorageDataSet* enables it to be called for all columns. See *ColumnChangeListener* for more information.

columnName

```
public final String getColumnName()
public final void setColumnName(String name)
```

Specifies the *columnName* used to identify the column in the source table. Use this property to access the data in the *DataSet* API. All column names used in the *DataSet* must be unique; JBuilder will change these names to make them unique as necessary.

For columns provided by a data source, the *columnName* property is set when the query is run. When a query is executed again, JDataStore uses this property to identify columns. If aliases are used, the *serverColumnName* property is also set.

If this property specifies a value other than the original column name on the server table, set the *serverColumnName* property so that JDataStore can resolve changes back to the data source.

The *DataSet* must be closed in order to set this property.

On error, the *setColumnName()* method throws a *DataSetException*.

See also *setCaption()*

columnPaintListener

```
public final ColumnPaintListener getColumnPaintListener()
```

The *ColumnPaintListener* interface is used for notification when painting of a value in a *Column* at a specific row location is occurring.

currency

```
public final boolean isCurrency()
public final void setCurrency(boolean currency)
```

Stores whether the *Column* defaults to currency formatting as specified in the locale for the *Column*. This property applies only to *Column* components containing numeric values. The default for this property is **false**, meaning that numeric data is not formatted as currency. To display values using default currency formatting, set this property to **true**.

An explicit *displayMask* always takes precedence over the formatting specified in the locale object. Specifying a display mask with currency symbols in it will display the value with currency formatting regardless of the *currency* setting.

See also *java.util.Locale*

dataSet

```
public StorageDataSet getDataSet()
```

Read-only property returns the *StorageDataSet* associated with this *Column* component.

dataType

```
public final int getDataType()
public final void setDataType(int dataType)
```

Stores the data type of the *Column*. Valid values for the data type are listed under *com.borland.dx.dataset.Variant* variables, except the *BTYE_ARRAY* variable. This property is used when saving changes to the local copy of the data in the *StorageDataSet* back to its original source. You typically do not change the data type of a column, however, the *DataSet* must be closed in order to set this property.

The *setDataType()* method can also be used to set the data type for a calculated field. It should not be called at run time when the *Column* already has data in it. On error, this method throws a *DataSetException*.

Warning Be careful when assigning data types to your columns. Coercions, whether intentional or not, can cause data loss. For example, coercion from a FLOAT data source to an INT column type discards the fractional part of the value. If you save the value back to the server, the INT is converted back to a FLOAT thereby losing the fractional part of the data.

For more information, see Data type conflicts between a column and its data source in the About section for this component.

default

```
public final String getDefault()
public final void setDefault(String defaultString)
```

Stores the default value for this *Column* in new records as a *String*.

The default value must be supplied in a locale-independent format, because the default may be coming from the server as metadata. This format is defined in the code comment for *Variant*'s *setFromString()* method. For example, a date must be in the format "yyyy-MM-dd".

If the *defaultString* parameter is set to "now" and the *dataType* for this *Column* is Date, Time, or Timestamp, the default will be the current time returned from *System.currentTimeMillis()*. The *setDefaultValue* method can be used to set the default as a *com.borland.dx.dataset.Variant*, which can be useful for binary data types.

Default values are automatically filled in when a new row is inserted. If no other updates are made to the row, it is considered untouched and is not posted.

defaultValue

```
public final Variant getDefaultValue()
public final void setDefaultValue(Variant defaultValue)
```

Stores the default value for fields in new records as a *com.borland.dx.dataset.Variant* of the same data type as the column.

displayMask

```
public final String getDisplayMask()
public final void setDisplayMask(String displayMask)
```

Stores the string specification used to format the data displayed in the *Column*. This can format a *Date* column to display as "MM/dd/yy" or "dd/MM/yy" for example. In numeric columns, this is how you show commas, decimal points, and currency symbols. In *String* columns, display masks can add spaces or special characters that are not contained in the data itself for display formatting purposes. For example, a telephone number stored as 4084311000 can be displayed as (408) 431-1000 using a display mask of "(999) 999-9999" (without quotes). A *displayMask* for a numeric column is always defined using commas for grouping and a period as a decimal separator. At run time, locale-appropriate characters are substituted.

The *displayMask* is also used to parse user input if no *editMask* is specified. It does not perform keystroke-by-keystroke validation (as with *editMasks*), however, it converts the entire *String* input back into the correct data type for the *Column*. User input which cannot be parsed using the specified display mask causes *JDataStore* to wire a message and keep the user in the edit field until the data entered is correct. When you want tighter character validation (for example, character by character validation), set the *editMask* property.

The edit mask has the same syntax as the *displayMask*, but it is used only for user input, not for display formatting.

There is no default display mask specification, however, explicitly setting this property to **null** or an empty string causes *JDataStore* to choose a display mask specification. If the data type of the column has international (localization) implications, the locale file for the *Column* is accessed for default display settings, for example, date and currency symbol. Otherwise, the *formatter* is bypassed and a basic display mask is used.

Because default display masks are locale-sensitive and built as needed, they are automatically localized. User-defined display masks are not altered if the column's *locale* changes.

Display masks and edit masks are never substituted if one is supplied and the other not. However, the display mask does perform validation of user input even if no edit mask is specified.

The *displayMask* property is an easy way to generate an instance of a *formatter* object which manages how the data is displayed. If you find that the *displayMask* does not handle your formatting needs, you can create your own class which implements the *ItemFormatStr* interface and assign it into the *Column* component's *formatter* property directly. In such cases, do not assign the *displayMask* property as your formatter will be overwritten.

To reset an edit mask back to its default, call the *setEditMask()* method.

Note When editing data formatted with a display or edit mask, the *Tab* and *Enter* keys attempt to post the field; they are not sub-field navigation keys. Pressing either key attempts to post the value to the *DataSet* and if incomplete, the cursor is positioned at the offending index. If a *StatusBar* or *JdbStatusLabel* control has its *dataSet* property set to the same *DataSet* and the error generated is a *ValidationException*, the error displays on the *StatusBar* or *JdbStatusLabel*. Otherwise, an error dialog displays with the error. To prevent this dialog from appearing, customize the *DataSetException* error handler to trap for the exception.

See also *editMask*, *exportDisplayMask*, String-based patterns (masks)

editable

```
public final boolean isEditable()
public final void setEditable(boolean editable)
```

Stores whether a *Column* is editable or not through a data-aware control. Programmatic edits to the *Column* are not affected by this property. If **false** (the default) a *Column* can be modified. To prevent a *Column* from being modified, set this property to **true**. To prevent edits to the *Column* data programmatically or through data-aware controls, set the *readOnly* property.

The *setEditable()* method throws a *DataSetException* when called on a *Column* of an open *DataSet*.

editMask

```
public final String getEditMask()
public void setEditMask(String editMask)
```

Stores the string specification used to format and control the entry of data in the column when the user starts editing data. Before editing starts, the *displayMask* handles the formatting and parsing. Edit masks affect the formatting of data by displaying spaces or special characters that ease data entry. For *String* columns, these characters may optionally be stored with the data or not. For example, displaying parenthesis for a telephone number can ease data entry by clearly separating the area code from the telephone number. An *editMask* for a numeric column is always defined using commas for grouping and a period as a decimal separator. At run time, locale-appropriate characters are substituted.

Edit masks also restrict the type of data the user can enter, for example, by allowing digits only, or requiring entry in certain parts of a field, and so on.

User-defined edit masks are not altered if the column's *locale* changes.

When entering data into a edit mask and using the left shift feature where characters are entered from the right end of the pattern specification and shift left, if any character input cannot shift left, shifting stops. If you continue to type when you are positioned at the right of the pattern, the last character is overwritten with each key pressed.

Edit and display masks are never substituted one for the other if one is supplied and the other not. However, if an edit mask is not specified, the display mask performs validation of user input.

If you find that the *editMask* does not handle your formatting needs, you can create your own class which implements the *ItemEditMask* interface and assign it into the *Column* component's *editMasker* property directly. In such cases, do not assign the *editMask* property as your *editMasker* will be overwritten.

To reset a display mask back to its default, call the *setDisplayMask()* method.

See also *displayMask* property, String-based patterns (masks)

editMasker

```
public final ItemEditMask getEditMasker()
public void setEditMasker(ItemEditMask editMasker)
```

The *ItemEditMask* implementation that provides the keystroke by keystroke validation of the data in the *Column*.

In most cases, the *editMask* property is used to provide formatting rules when data in the *Column* is being entered or edited. From the *editMask*, an *editMasker* object is created that manages the validation of the data entry or data editing.

If you want custom rules beyond those provided by the *com.borland* classes, create your own class that implements *ItemEditMask* and set this property to your custom class. Do not set the *editMask* property in addition to the *editMasker* property as the *editMasker* will be overridden.

exportDisplayMask

```
public final String getExportDisplayMask()
public final void setExportDisplayMask(String exportDisplayMask)
```

The string display mask specification that is used when importing and exporting data to a text file.

When exporting, *JDataStore* creates a *schema* file (.SCHEMA file extension) that contains information about the data being exported so that it can be easily read back into a *TableDataSet* component. This mask is also written into the schema file. The *exportDisplayMask* value in the schema file always takes precedence—you cannot override this value by setting a new *exportDisplayMask*. A *displayMask* for a numeric column is always defined using commas for grouping and a period as a decimal separator. At run time, locale-appropriate characters are substituted.

Data can be imported into *JDataStore* in two ways: with a valid schema file, or by specifying column names and data types. When data is exported by *JDataStore*, a schema file is created to define the data. When importing data that has been previously exported by *JDataStore*, assuming that the schema file was created, *JDataStore* uses the *exportDisplayMask* information contained in the file to import the data. If the schema file doesn't exist (meaning that the data has not yet been exported by *JDataStore*), the setting for this property is used to import the data. If there is no schema file and the *exportDisplayMask* property is not set, and the data type of the column has international (localization) implications, the locale file for the *Column* is accessed for default display settings, for example, date and currency symbol. Otherwise, the export formatter is bypassed and a basic export mask is used.

The *exportDisplayMask* specification is used to create an instance of an *exportFormatter* (a *VariantFormatter*) class object that controls the formatting of the exported data. If you want custom rules beyond those provided by the *com.borland* classes, create your own class that implements *ItemFormatStr* and set the *exportFormatter* property to your custom class. In this case, do not set the *exportDisplayMask* as it will overwrite the *exportFormattter* property.

To reset an export display mask back to its default, call the *setExportDisplayMask()* method.

See also *displayMask* property, "String-based patterns (masks)"

exportFormatter

```
public final VariantFormatter getExportFormatter()
public final void setExportFormatter(VariantFormatter exportFormatter)
```

The *VariantFormatter* implementation that provides formatting rules when exporting data in the *Column*. The implementation is specific to the data type of the *Column*, for example, *BigDecimalFormatter*.

In most cases, the *exportDisplayMask* property is used to provide formatting rules when exporting the data in a *Column*. From the *exportDisplayMask*, an *exportFormatter* object is created that manages what the data looks like when it is exported.

If you want custom formatting rules beyond those provided by the *com.borland* classes, create a custom class that extends *ItemFormatStr* and set this property to an instance of that class. Do not set the *exportDisplayMask* property in addition to the *exportFormatter* property as the *exportFormatter* will be overridden.

fixedPrecision

```
public boolean isFixedPrecision()
public void setFixedPrecision(boolean fixedPrecision)
```

Stores whether the precision of decimal point values is fixed (**true**) or not (**false**).

font

```
public final Font getFont()
public final void setFont(Font font)
```

Stores the font used to display the *Column* in a data-aware control.

foreground

```
public final Color getForeground()
public final void setForeground(Color foreground)
```

Stores the foreground color of the *Column* when displayed in a data-aware control. Any color object is valid. The *java.awt.color* provides class variables for some common colors.

formatter

```
public final VariantFormatter getFormatter()
public final void setFormatter(VariantFormatter formatter)
```

The *VariantFormatter* implementation that provides formatting rules when displaying data in the *Column*. The implementation is specific to the data type of the *Column*, for example, *BigDecimalFormatter*.

In most cases, the *displayMask* property is used to provide formatting rules for a *Column*. From the *displayMask*, a *formatter* object is created that manages the display of the data.

If you want more custom display rules than are provided by the *com.borland* classes, create a custom class that extends the *VariantFormatter* (or one of its subclasses) and set this property to an instance of that class. Do not set a *displayMask* in addition to the *formatter* property as the *formatter* will be overridden.

hidden

```
public final boolean isHidden()
public final void setHidden(boolean hidden)
```

This method is used internally by other *com.borland* classes. Do not use this method directly.

itemEditor

```
public final Object getItemEditor()
public final void setItemEditor(Object itemEditor)
```

Sets an optional Swing *CellEditor* for use by a visual component bound to the *Column* when editing.

itemPainter

```
public final Object getItemPainter()
public final void setItemPainter(Object itemPainter)
```

Sets an optional Swing *CellRenderer* for use by a visual component bound to the *Column* when painting.

javaClass

```
public final String getJavaClass()
public final void setJavaClass(String className)
```

Sets the *javaClass*.

locale

```
public final Locale getLocale()
public final void setLocale(Locale locale)
```

Stores the *Locale* object to use for formatting a *Column*. Locale contains country or area-specific formatting specifications such as date format (MM/DD/YY, DD/MM/YY, YY/MM/DD), currency symbol, and so on.

If this property is not specified at the *Column* level, it defaults to the *DataSet* component's *locale*, if specified. Otherwise it defaults to the *locale* of the Java environment.

max

```
public final String getMax()
public final void setMax(String maxString)
```

Specifies the maximum allowable value (inclusive) in this *Column* as a *String*. The *setMax()* method throws a *DataSetException* if called on a *Column* of an open *DataSet*.

The maximum value must be supplied in a locale-independent format, because the maximum may be coming from the server as metadata. This format is defined in the code comment for *Variant*'s *setFromString()* method. For example, a date must be in the format "yyyy-MM-dd".

maxInline

```
public final int getMaxInline()
public final void setMaxInline(int maxInline)
```

This property can be used on columns of a *StorageDataSet* that have their *store* property set to a *DataStore*. This property is ignored by columns of a *StorageDataSet* that has its *store* property set to **null** or *MemoryStore*.

This property specifies the maximal "inline" length, in bytes, for Strings and InputStreams. If a *Column* value is stored with a length that exceeds this setting, it is stored as a BLOB, which is a little less efficient. To improve performance, set *maxInline* to a higher value. Note that setting *maxInline* to a higher value limits the maximum number of fields that can be stored in a row.

maxValue

```
public final Variant getMaxValue()
public final void setMaxValue(Variant maxValue)
```

Specifies the maximum value allowable (inclusive) in this *Column* as a *Variant*. The *setMaxValue()* method throws a *DataSetException* if called on a *Column* of an open *DataSet*.

min

```
public final String getMin()
public final void setMin(String minString)
```

Specifies the minimum allowable value (inclusive) in this *Column*. The *setMin()* method throws a *DataSetException* if called on a *Column* of an open *DataSet*.

The minimum value must be supplied in a locale-independent format, because the minimum may be coming from the server as metadata. This format is defined in the code comment for *Variant*'s *setFromString()* method. For example, a date must be in the format "yyyy-MM-dd".

minValue

```
public final Variant getMinValue()
public final void setMinValue(Variant minValue)
```

Stores the minimum value allowable (inclusive) in this *Column* as a *Variant*. The *setMinValue()* method throws a *DataSetException* if called on a *Column* of an open *DataSet*.

ordinal

```
public int getOrdinal()
```

Read-only property that specifies the ordinal position of the *Column* component within the *DataSet*.

parameterType

```
public int getParameterType()
public void setParameterType(int parameterType)
```

Specifies the type of parameter in a *ParameterRow*. For valid values for this property, see *ParameterType* variables.

persist

```
public final boolean isPersist()
public final void setPersist(boolean persist)
```

Stores whether the *Column* is persisted in a *DataSet* when the application is run.

By default, the columns that display in a data-aware control are determined at run-time based on the *Columns* that appear in the *DataSet*. If your application depends on particular columns displaying in the control, set this property programmatically to **true** before the data is provided. If the source column of the persistent column changes or is deleted, the column is left empty. No *Exception* is thrown.

Calling *StorageDataSet.setColumns(...)* sets the *persist* property to true for all columns passed in.

For information on how column persistence can help when a master-detail relationship might have an empty master *DataSet*, see "Empty master *DataSet*" in the About section of the *MasterLinkDescriptor* class.

pickList

```
public final PickListDescriptor getPickList()
public final void setPickList(PickListDescriptor pickList)
```

Specifies the *PickListDescriptor* for the *Column* which describes the relationship between the *Column* and a second, separate “pick list” or “lookup” *DataSet*.

The *DataSet* must be closed in order to set this property.

For more information on pick lists, see the *PickListDescriptor* class.

precision

```
public final int getPrecision()
public final void setPrecision(int precision)
```

Stores the precision used for the *Column* in data-aware controls. For a *String* column, precision represents the maximum length of a value.

preferredOrdinal

```
public int getPreferredOrdinal()
public void setPreferredOrdinal(int preferredOrdinal)
```

Stores the preferred ordinal position of the *Column* in the *DataSet*. Normally, Columns that are defined in your application that are not provided by the query or procedure will appear before columns that are provided. Set this property to any value greater than or equal to zero to move this *Column* to that position in the *DataSet*. The first column ordinal is zero. You can set more than one column to the same ordinal value, however, the results may not be predictable. Columns whose *preferredOrdinal* is not set retain their position relative to each other.

This property defaults to -1 meaning that there is no preferred position for this *Column*.

The *DataSet* must be closed in order to set this property.

readOnly

```
public final boolean isReadOnly()
public final void setReadOnly(boolean readOnly)
```

Stores whether the *Column* is writable or read-only. Setting *readOnly* to **true** disables modification of the data in the *Column* (both programmatically and through a UI control) and prevents the user from navigating to the *Column*. Setting *readOnly* to **false** allows both modification of the data and navigation to the *Column*. To restrict data editing from a data-aware control only (but allow data editing programmatically), set the *editable* property of the *Column*.

This property is automatically set to **true** when you set the *calcType* of this *Column* to any value other than *NO_CALC*.

A *DataSetException* is thrown if the *setReadOnly()* method is called on a *Column* of an open *DataSet*.

required

```
public final boolean isRequired()
public final void setRequired(boolean required)
```

Specifies whether values in the *Column* may be left blank at the time of posting the row of data to the *DataSet*. A *DataSetException* is thrown if the *setRequired()* method is called on a *Column* of an open *DataSet*.

resolvable

```
public boolean isResolvable()
public void setResolvable(boolean resolvable)
```

Specifies whether the *Column* is included when resolving changes back to its data source. By default, all non-calculated columns are set to **true**. If set to **false**, this *Column* will not be saved back to its data source during the resolution. This property is useful when saving a *DataSet* back to a SQL server when the *DataSet* has calculated or aggregation columns that should not be included in the resolution.

rowId

```
public final boolean isRowId()
public final void setRowId(boolean rowId)
```

Specifies whether the *Column* contains data that uniquely identifies the row. A unique row identifier (also known as a unique *rowID*) is necessary to save changes to the data in the *DataSet* back to its data source.

Several columns may have their *rowId* property set to **true**. This indicates that the values in these columns together uniquely identify a row.

Note The *setRowId(...)* method does not reset the edit blocked state of the *StorageDataSet*. Also, if *JDataStore* has flagged that a *DataSet* is not updateable because it has no row IDs, this flag will be removed when you define row IDs.

scale

```
public final int getScale()
public final void setScale(int scale)
```

Specifies the number of digits to the right of the decimal point for BIGDECIMAL values. This property is used by data-aware controls when displaying values in the *Column*.

You should set this property when working with BIGDECIMAL values. Otherwise, you get far more decimal places than most applications need.

schemaName

```
public final String getSchemaName()
public final void setSchemaName(String schemaName)
```

Specifies the schema name of the table that the *Column* belongs to. Used mostly with *QueryDataSets* to flag the table that a *Column* belongs to. This property may be left **null**.

searchable

```
public boolean isSearchable()
public void setSearchable(boolean searchable)
```

Specifies whether a *Column* is searchable (**true**) or not (**false**). Not all servers allow search operations on all data types, for example, binary data is commonly not searchable. *JDataStore* attempts to read this information from the server metadata whenever possible.

Columns that are not searchable are not used in the comparisons of an update query, even when the *updateMode* property is set to *ALL_COLUMNS*.

Note Not all data accessed from SQL servers are searchable. Check your server documentation for more information on capabilities supported on its data types.

serverColumnName

```
public final String getServerColumnName()
public final void setServerColumnName(String serverColumnName)
```

Specifies the column name used by the *QueryResolver* when generating resolution queries. This property should correspond to a name of a physical column in the table of the database. *JDataStore* uses this name to save back changes to the table on the server. If null, the *columnName* property is used instead.

sortable

```
public final boolean isSortable()
```

Read-only property that stores whether the *Column* is sortable or not. *JDataStore* attempts to read such information from the server whenever possible. Typically, all *Column* components are sortable except those containing binary or Object data.

sortPrecision

```
public final int getSortPrecision()
public final void setSortPrecision(int sortPrecision)
```

Stores the precision of the data in this *Column*. This property is used only for non english locales that use Collation keys for sorting.

sqlType

```
public final int getSqlType()
public void setSqlType(int sqlType)
```

Stores the type of JDBC *SqlType* of the *Column*. Accepted values for *SqlType* are defined in *java.sql.Types*.

tableName

```
public final String getTableName()
public final void setTableName(String tableName)
```

Stores the *String* table name that the *Column* belongs to. This property is used by the *QueryDataSet* component to identify the table a *Column* belongs to. For all *StorageDataSet* objects, the table name is the same for all *Column* components (unless the *Column* components were instantiated as a result of a SQL JOIN statement), or **null**.

textual

```
public final boolean isTextual()
```

Stores whether the *Column* contains data that can be represented by a *String*. For example, numeric, date, and string data return **true**. Data stored as binary data return **false**.

visible

```
public final int getVisible()
public final void setVisible(int visible)
```

Specifies whether the *Column* is visible or hidden. If not set, default logic is used to determine whether to display the *Column*. For example, the linking column(s) of the detail *DataSet* in a master-detail relationship are hidden by default. Accepted values for *visible* are listed in *com.borland.jb.util.TriStateProperty*.

width

```
public int getWidth()
public void setWidth(int width)
```

The number of characters requested to display in a control for the given *Column*. The number of characters that can be entered into the column is not restricted by this property, only the display. This property is used in data-aware controls when displaying the *Column*.

For information on setting the width of a *Column* component's data (for example, the width of a *String* column), see the *precision* property.

Column methods

Method	Implemented in
<code>clone()</code>	this class
<code>copy()</code>	this class
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>format(com.borland.dx.dataset.Variant)</code>	this class
<code>getDefault(com.borland.dx.dataset.Variant)</code>	this class
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>hasValidations()</code>	this class
<code>notify()</code>	<code>java.lang.Object</code>
<code>notifyAll()</code>	<code>java.lang.Object</code>
<code>setDisplayMask()</code>	this class
<code>setEditMask()</code>	this class
<code>setExportDisplayMask()</code>	this class
<code>toString()</code>	this class
<code>validate(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Variant)</code>	this class
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>

clone()

`public Object clone()`

Clones the *Column* and returns it in *Object*.

Overrides `java.lang.Object.clone()`

copy()

`public Column copy()`

Creates and returns a complete copy of the *Column*, without unbinding the *StorageDataSet*.

format(com.borland.dx.dataset.Variant)

`public final String format(Variant value)`

Uses the *formatter* property to produce a *String* representation of the *Variant*. If there is no formatter for this *Column*, *value.toString()* is returned.

value The *Variant* value to return formatted as a *String*.

getDefault(com.borland.dx.dataset.Variant)

```
public final void getDefault(Variant value)
```

Returns the default value set for the *Column*.

hasValidations()

```
public boolean hasValidations()
```

Read-only property that returns whether the *Column* has specified validation criteria in the form of:

- readOnly
- a minimum value
- a maximum value
- for a *String* column, precision greater than 0
- a registered *ColumnChangeListener*

If any of the above apply, this method returns **true**. Also, calculated and aggregate columns return **true**. Otherwise this method returns **false**.

setDisplayMask()

```
public final void setDisplayMask()
```

Sets the *Column* to its default *formatter*. This is different from calling *setDisplayMask(null)*, which explicitly asks to select a preferred *formatter* based on *locale*.

setEditMask()

```
public final void setEditMask()
```

Sets the *Column* to its default *editMasker*. This is different from calling *setEditMask(null)*, which explicitly asks to select a preferred *editMasker* based on *locale*.

setExportDisplayMask()

```
public final void setExportDisplayMask()
```

Sets the *Column* to its default formatter for import/export. This is different from *setExportDisplayMask(null)*, which explicitly asks to select a preferred formatter based on *locale*.

toString()

```
public String toString()
```

Returns the *String* representation of the *Column*.

Overrides `java.lang.Object.toString()`

validate(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Variant)

```
public void validate(DataSet editDataSet, Variant value)
```

Validates the specified *value* against the *Column* component's validation criteria (minimum value, maximum value, and so on). On error, this method throws a *ValidationException* or *DataSetException* as applicable.

Column event listeners

This component is a source for the following event sets.

coerceFrom

```
public void addCoerceFromListener(CoerceFromListener listener)
public synchronized void removeCoerceFromListener(CoerceFromListener listener)
```

coerceTo

```
public void addCoerceToListener(CoerceToListener listener)
public synchronized void removeCoerceToListener(CoerceToListener listener)
```

columnChange

```
public void addColumnChangeListener(ColumnChangeListener listener)
public synchronized void removeColumnChangeListener(ColumnChangeListener listener)
```

columnPaint

```
public void addColumnPaintListener(ColumnPaintListener listener)
public synchronized void removeColumnPaintListener(ColumnPaintListener listener)
```

ColumnAware interface

dx.dataset package

Extends `com.borland.dx.dataset.DataSetAware`

The *ColumnAware* interface is a way for a component to declare to *JDataStore* that it knows how to bind to a specific column in a specific *DataSet*. Note that

the interface's methods defined here are valid JavaBeans property getters/setters by design to simplify the component's implementation.

The *ColumnAware* interface extends the *DataSetAware* interface. If your control does not need to identify a specific column of data (a grid control, for example), implement the *DataSetAware* interface instead.

ColumnAware properties

Property	Implemented in
columnName	this class
dataSet	com.borland.dx.dataset.DataSetAware

columnName

```
public String getColumnName()  
public void setColumnName(String columnName)
```

Determines which *Column* is accessed by the control that implements this interface in the *DataSet*.

ColumnChangeAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.ColumnChangeListener, java.util.EventListener

This class is an adapter class for *ColumnChangeListener*, which is used for notification when a field value changes.

ColumnChangeAdapter properties

Property	Implemented in
class*	java.lang.Object

ColumnChangeAdapter methods

Method	Implemented in
changed(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant)	this class
clone()	java.lang.Object

Method	Implemented in
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>notify()</code>	<code>java.lang.Object</code>
<code>notifyAll()</code>	<code>java.lang.Object</code>
<code>toString()</code>	<code>java.lang.Object</code>
<code>validate(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant)</code>	this class
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>

ColumnChangeListener interface

dx.dataset package

Extends `java.util.EventListener`

Implemented by `com.borland.dx.dataset.ColumnChangeAdapter`

This interface is used for notification that data has changed in a column. One listener can be written for each column, or for an entire *StorageDataSet*, for which you want to be notified of changes. This is the listener to use for checks on field values that should be done before a user leaves the field. One example of when this might be useful is to check that a part number is in stock before the rest of the line item information is entered. Writing a listener for a *StorageDataSet* enables it to be called for all columns.

ColumnChangeListener methods

Method	Implemented in
<code>changed(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant)</code>	this class
<code>validate(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant)</code>	this class

**changed(`com.borland.dx.dataset.DataSet`,
`com.borland.dx.dataset.Column`, `com.borland.dx.dataset.Variant`)**

`void changed(DataSet dataSet, Column column, Variant value)`

Called after all column level validations have been performed and a column value has been successfully posted inside a *ReadWriteRow*.

<i>dataSet</i>	Which data set contains the column that has data that has been accepted as a valid field value.
<i>column</i>	Which column has data that has been changed and verified as valid.
<i>value</i>	The new, valid value of the data in the column that changed.

**validate(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.Column`,
`com.borland.dx.dataset.Variant`)**

`void validate(DataSet dataSet, Column column, Variant value)`

Called before column level validations like *readOnly*, *min*, or *max* are performed, and before the new value is recorded in a *ReadWriteRow*. Note that if values are set programmatically, *EditMask* constraints are not applied. You can change the value stored in *value*, but it must still pass the column level validations. To prevent the value from being set, throw an *Exception*. If an *Exception* is constructed with a *String* parameter, this *String* is used in the default error handling display, for example,

```
throw new Exception("My error message");
```

<i>dataSet</i>	Which data set contains the column that has data that has changed.
<i>column</i>	Which column has data that has changed.
<i>value</i>	The new value of the data in the column that changed.

ColumnList component

`dx.dataset` package

Extends `java.lang.Object`

The *ColumnList* component contains a high speed database column name lookup mechanism. In this system, *Column* names are cached as immutable Java String “references”. Once a reference is added to the cache, the value that the reference points to can never change. This is because it is immutable and because memory in the Java environment is only freed when no active

objects are referencing it (Java uses a garbage collection memory management scheme).

In such an environment, the process of looking up a name can be sped up dramatically by caching the reference. Once a String name is cached, future lookups can be performed by very efficient String object reference comparisons. No comparison of the String value is necessary once the String has been added to the cache.

If a String name is not found in the reference cache, a (more expensive) hash lookup by value is performed. Once found, the String name is added to the reference cache.

The reference cache is “least recently allocated”. This means that the least recently allocated entry is reused when a new item needs to be added. The cache is not stored as an array of entries because inline compares of cache values is much faster (String arrays are range checked and have the overhead of array subscripting).

ColumnList constructors

ColumnList()

```
public ColumnList()
```

Creates a *ColumnList* component. On error this constructor throws a *DataSetException*.

ColumnList(com.borland.dx.dataset.StorageDataSet)

```
public ColumnList(StorageDataSet dataSet)
```

Creates a *ColumnList* component using the columns in the specified *StorageDataSet*. On error this constructor throws a *DataSetException*.

ColumnList(int)

```
public ColumnList(int size)
```

Creates a *ColumnList* component using the specified number of entries. On error this constructor throws a *DataSetException*.

ColumnList properties

Property	Implemented in
class*	java.lang.Object
columnsArray*	this class

columnsArray

public final Column[] getColumnArray()

Read-only property that returns the array of *Column* components in the *ColumnList*.

ColumnList methods

Method	Implemented in
addColumn(com.borland.dx.dataset.Column)	this class
addColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, boolean, boolean)	this class
clone()	java.lang.Object
cloneColumns()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findOrdinal(java.lang.String)	this class
getColumn(java.lang.String)	this class
hasColumn(java.lang.String)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

addColumn(com.borland.dx.dataset.Column)

public final int addColumn(Column column)

Adds the specified *Column* to this component. If the *column* parameter is **null**, this method throws a *DataSetException* of *NULL_COLUMN_NAME*.

addColumn(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, boolean, boolean)

public final int addColumn(StorageDataSet dataSet, Column column, boolean isUnique, boolean setOrdinals)

Adds the specified *Column* in the specified *DataSet* to this component. If the *column* parameter is **null**, this method throws a *DataSetException* of *NULL_COLUMN_NAME*.

isUnique is a constraint on column values. *setOrdinals* is a boolean value which sets the index into the array of *Columns*.

cloneColumns()

```
public final Column[] cloneColumns()
```

Returns an array of columns which are a copy of the *ColumnList*.

findOrdinal(java.lang.String)

```
public final int findOrdinal(String columnName)
```

Returns the ordinal of the column specified in *columnName*.

getColumn(java.lang.String)

```
public final Column getColumn(String columnName)
```

Returns the *Column* component given its *columnName*. On error, this method throws a *DataSetException*.

hasColumn(java.lang.String)

```
public final Column hasColumn(String columnName)
```

Checks to see if the specified *columnName* is in the *ColumnList*. If the *columnName* parameter isn't specified or the *columnName* isn't in the *ColumnList*, this method returns **null**. If it is in the *ColumnList*, this method returns the *Column* component.

ColumnPaintAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.ColumnPaintListener, java.util.EventListener

This class is an adapter class for the *ColumnPaintListener*, which is used for *Column* paint events.

ColumnPaintAdapter properties

Property	Implemented in
class*	java.lang.Object

ColumnPaintAdapter methods

Method	Implemented in
clone()	java.lang.Object
editing(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.CustomPaintSite)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
painting(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, int, com.borland.dx.dataset.Variant, com.borland.dx.dataset.CustomPaintSite)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ColumnPaintListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.ColumnPaintAdapter

The *ColumnPaintListener* interface is used for notification when painting of a value in a *Column* at a specific row location is occurring.

ColumnPaintListener methods

Method	Implemented in
editing(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.CustomPaintSite)	this class
painting(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, int, com.borland.dx.dataset.Variant, com.borland.dx.dataset.CustomPaintSite)	this class

editing(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.Column`, `com.borland.dx.dataset.CustomPaintSite`)

`void editing(DataSet dataSet, Column column, CustomPaintSite paintSite)`

Use this method to set display properties for the editor on the incoming *paintSite*.

painting(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.Column`, `int`, `com.borland.dx.dataset.Variant`, `com.borland.dx.dataset.CustomPaintSite`)

`void painting(DataSet dataSet, Column column, int row, Variant value, CustomPaintSite paintSite)`

Use this method to set display properties on the incoming *paintSite* and/or modify the incoming value. If *value* is modified, it does not affect the value stored in the *dataSet*. It only affects the value to be displayed.

ColumnVariant class

`dx.dataset` package

Extends `com.borland.dx.dataset.Variant`

Implements `java.io.Serializable`, `java.lang.Cloneable`

This class extends the *com.borland.dx.dataset.Variant* class by adding two methods to retrieve *DataSet* and *Column* information associated with this value.

Visual components that receive values from a *DataSet* will receive them as *ColumnVariant*. These components can test the Object value received in order to collect more contextual information about the value.

ColumnVariant variables

Variable	Defined in
ASSIGNED_NULL	<code>com.borland.dx.dataset.Variant</code>
AssignedNull_S	<code>com.borland.dx.dataset.Variant</code>
BIGDECIMAL	<code>com.borland.dx.dataset.Variant</code>
BigDecimalType_S	<code>com.borland.dx.dataset.Variant</code>
BinaryStreamType_S	<code>com.borland.dx.dataset.Variant</code>
BOOLEAN	<code>com.borland.dx.dataset.Variant</code>
BooleanType_S	<code>com.borland.dx.dataset.Variant</code>
BYTE	<code>com.borland.dx.dataset.Variant</code>
BYTE_ARRAY	<code>com.borland.dx.dataset.Variant</code>

Variable	Defined in
ByteArrayType_S	com.borland.dx.dataset.Variant
ByteType_S	com.borland.dx.dataset.Variant
DATE	com.borland.dx.dataset.Variant
DateType_S	com.borland.dx.dataset.Variant
DOUBLE	com.borland.dx.dataset.Variant
DoubleType_S	com.borland.dx.dataset.Variant
FLOAT	com.borland.dx.dataset.Variant
FloatType_S	com.borland.dx.dataset.Variant
INPUTSTREAM	com.borland.dx.dataset.Variant
InputStreamType_S	com.borland.dx.dataset.Variant
INT	com.borland.dx.dataset.Variant
IntType_S	com.borland.dx.dataset.Variant
LONG	com.borland.dx.dataset.Variant
LongType_S	com.borland.dx.dataset.Variant
MaxTypes	com.borland.dx.dataset.Variant
NULL_TYPES	com.borland.dx.dataset.Variant
nullVariant	com.borland.dx.dataset.Variant
OBJECT	com.borland.dx.dataset.Variant
ObjectType_S	com.borland.dx.dataset.Variant
SHORT	com.borland.dx.dataset.Variant
ShortType_S	com.borland.dx.dataset.Variant
STRING	com.borland.dx.dataset.Variant
StringType_S	com.borland.dx.dataset.Variant
TIME	com.borland.dx.dataset.Variant
TIMESTAMP	com.borland.dx.dataset.Variant
TimestampType_S	com.borland.dx.dataset.Variant
TimeType_S	com.borland.dx.dataset.Variant
UNASSIGNED_NULL	com.borland.dx.dataset.Variant
UnassignedNull_S	com.borland.dx.dataset.Variant
UnknownType_S	com.borland.dx.dataset.Variant

ColumnVariant constructors

ColumnVariant(com.borland.dx.dataset.Column, com.borland.dx.dataset.DataSet)

public ColumnVariant(Column column, DataSet dataSet)

Constructs a *ColumnVariant* object with the specified parameters.

- column* The *Column* component that contains the *Variant*.
- dataSet* The *DataSet* that contains the *Variant*.

ColumnVariant properties

Property	Implemented in
arrayLength	com.borland.dx.dataset.Variant
asBigDecimal*	com.borland.dx.dataset.Variant
asBoolean*	com.borland.dx.dataset.Variant
asDate**	com.borland.dx.dataset.Variant
asDouble*	com.borland.dx.dataset.Variant
asFloat*	com.borland.dx.dataset.Variant
asInt*	com.borland.dx.dataset.Variant
asLong*	com.borland.dx.dataset.Variant
asObject*	com.borland.dx.dataset.Variant
asShort*	com.borland.dx.dataset.Variant
assignedNull*	com.borland.dx.dataset.Variant
asTime**	com.borland.dx.dataset.Variant
asTimestamp**	com.borland.dx.dataset.Variant
asVariant**	com.borland.dx.dataset.Variant
bigDecimal	com.borland.dx.dataset.Variant
binaryStream	com.borland.dx.dataset.Variant
boolean	com.borland.dx.dataset.Variant
byte	com.borland.dx.dataset.Variant
byteArray*	com.borland.dx.dataset.Variant
class*	java.lang.Object
column*	this class
dataSet*	this class
date	com.borland.dx.dataset.Variant
displayValue*	com.borland.dx.dataset.Variant
double	com.borland.dx.dataset.Variant
float	com.borland.dx.dataset.Variant
inputStream	com.borland.dx.dataset.Variant
int	com.borland.dx.dataset.Variant
long	com.borland.dx.dataset.Variant
null	com.borland.dx.dataset.Variant
object	com.borland.dx.dataset.Variant
setType*	com.borland.dx.dataset.Variant
short	com.borland.dx.dataset.Variant
string	com.borland.dx.dataset.Variant
time	com.borland.dx.dataset.Variant
timestamp	com.borland.dx.dataset.Variant
type*	com.borland.dx.dataset.Variant

Property	Implemented in
unassignedNull*	com.borland.dx.dataset.Variant
variant**	com.borland.dx.dataset.Variant

column

public Column getColumn()
Read-only property that returns the *Column* component that contains the *Variant*.

dataSet

public DataSet getDataSet()
Read-only property that returns the *DataSet* that contains the *Variant*.

ColumnVariant methods

Method	Implemented in
add(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.Variant
clone()	com.borland.dx.dataset.Variant
compareTo(com.borland.dx.dataset.Variant)	com.borland.dx.dataset.Variant
currentUTCTimeMillis()	com.borland.dx.dataset.Variant
equals(com.borland.dx.dataset.Variant)	com.borland.dx.dataset.Variant
equals(java.lang.Object)	java.lang.Object
equalsInstance(com.borland.dx.dataset.Variant)	com.borland.dx.dataset.Variant
finalize()	java.lang.Object
getTimeZoneOffset()	com.borland.dx.dataset.Variant
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
setAsObject(java.lang.Object, int)	com.borland.dx.dataset.Variant
setAssignedNull()	com.borland.dx.dataset.Variant
setByteArray(byte[], int)	com.borland.dx.dataset.Variant
setFromString(int, java.lang.String)	com.borland.dx.dataset.Variant
setTimestamp(long, int)	com.borland.dx.dataset.Variant
setUnassignedNull()	com.borland.dx.dataset.Variant
subtract(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.Variant
toString()	com.borland.dx.dataset.Variant
typeId(java.lang.String)	com.borland.dx.dataset.Variant

Method	Implemented in
typeName(int)	com.borland.dx.dataset.Variant
typeOf(java.lang.String)	com.borland.dx.dataset.Variant
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

CountAggOperator class

dx.dataset package

Extends com.borland.dx.dataset.AggOperator

Implements java.io.Serializable, java.lang.Cloneable

The *CountAggOperator* class is instantiable subclass of the *AggOperator* and defines an aggregation operation of a count.

Set the *aggOperator* property of the *AggDescriptor* object to this class to perform a count calculation using the property settings stored in the *AggDescriptor*. Attach the *AggDescriptor* object to a *Column* component's *agg* property to access the data that the aggregation uses.

CountAggOperator variables

Variable	Defined in
aggColumn	com.borland.dx.dataset.AggOperator
aggDataSet	com.borland.dx.dataset.AggOperator
aggValue	com.borland.dx.dataset.AggOperator
dataSet	com.borland.dx.dataset.AggOperator
resultColumn	com.borland.dx.dataset.AggOperator
resultValue	com.borland.dx.dataset.AggOperator

CountAggOperator properties

Property	Implemented in
class*	java.lang.Object

CountAggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	this class
clone()	com.borland.dx.dataset.AggOperator
delete(com.borland.dx.dataset.ReadRow, long)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	this class
locate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.AggOperator
needsAggDataSet()	com.borland.dx.dataset.AggOperator
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.AggOperator
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add(com.borland.dx.dataset.ReadRow, long, boolean)

public void add(ReadRow row, long internalRow, boolean first)

A row has been added or updated.

- row*

The row containing the values.
- internalRow*

The unique identifier for the row.
- first*

Returns **true** if this is the first row in the group, **false** otherwise.

Overrides com.borland.dx.dataset.AggOperator.add(com.borland.dx.dataset.ReadRow, long, boolean)

delete(com.borland.dx.dataset.ReadRow, long)

```
public void delete(ReadRow row, long internalRow)
```

A row has been deleted or updated. The *ReadRow* parameter contains the values and *internalRow* is a unique identifier for the row.

Overrides `com.borland.dx.dataset.AggOperator.delete(com.borland.dx.dataset.ReadRow, long)`

init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)

```
public void init(StorageDataSet dataSet, String[] groupColumnNames, StorageDataSet aggDataSet,
    Column resultColumn, Column aggColumn)
```

Initializes the *CountAggOperator* by calling the constructor of its superclass.

<i>dataSet</i>	The <i>StorageDataSet</i> that contains the <i>aggColumn</i> that is being aggregated on
<i>groupColumnNames</i>	A array of the <i>Column</i> names to perform grouping by.
<i>aggDataSet</i>	The internal <i>StorageDataSet</i> that contains and maintains aggregated values.
<i>resultColumn</i>	The <i>Column</i> in <i>aggDataSet</i> that contains the aggregated value.
<i>aggColumn</i>	The <i>Column</i> in the <i>DataSet</i> that is being aggregated on.

Overrides `com.borland.dx.dataset.AggOperator.init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)`

CustomAggOperator class

dx.dataset package

Extends `com.borland.dx.dataset.AggOperator`

Implements `java.io.Serializable, java.lang.Cloneable`

The *CustomAggOperator* class is used internally as a place holder for *Column* values that are aggregated by using the *StorageDataSet.calcAggFieldsListener* event.

- To aggregate *Column* values with the *StorageDataSet.calcAggFieldsListener* event, you must:
- 1 Set the *aggColumn* in the *Column*'s *AggDescriptor* property to **null**.
 - 2 Set the *aggOperator* in the *Column*'s *AggDescriptor* property to **null**.
 - 3 Add an aggregate *Column* to maintain the aggregate value. Set its *calcType* property to *calcType.Aggregate*.
 - 4 Register a *CalcAggFieldsListener* with the aggregate *Column*'s *StorageDataSet*.

CustomAggOperator variables

Variable	Defined in
aggColumn	com.borland.dx.dataset.AggOperator
aggDataSet	com.borland.dx.dataset.AggOperator
aggValue	com.borland.dx.dataset.AggOperator
dataSet	com.borland.dx.dataset.AggOperator
resultColumn	com.borland.dx.dataset.AggOperator
resultValue	com.borland.dx.dataset.AggOperator

CustomAggOperator properties

Property	Implemented in
class*	java.lang.Object
updatable*	this class

updatable

public final boolean isUpdatable()

This property is used internally by other *com.borland* classes. You should never use this property directly.

CustomAggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	this class
clone()	com.borland.dx.dataset.AggOperator
delete(com.borland.dx.dataset.ReadRow, long)	this class
equals(java.lang.Object)	java.lang.Object

Method	Implemented in
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	com.borland.dx.dataset.AggregateOperator
locate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.AggregateOperator
needsAggregateDataSet()	com.borland.dx.dataset.AggregateOperator
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.AggregateOperator
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add(com.borland.dx.dataset.ReadRow, long, boolean)

public void add(ReadRow row, long internalRow, boolean first)

This method is used internally by other *com.borland* classes. You should never use this method directly. This method is a “dummy” add request. The *StorageDataSet.calcAggregateFieldsListener* maintains the aggregate.

Overrides com.borland.dx.dataset.AggregateOperator.add(com.borland.dx.dataset.ReadRow, long, boolean)

delete(com.borland.dx.dataset.ReadRow, long)

public void delete(ReadRow row, long internalRow)

This method is used internally by other *com.borland* classes. You should never use this method directly. This method is a “dummy” delete request. The *StorageDataSet.calcAggregateFieldsListener* maintains the aggregate.

Overrides com.borland.dx.dataset.AggregateOperator.delete(com.borland.dx.dataset.ReadRow, long)

CustomPaintSite interface

dx.dataset package

The *ItemPaintSite* interface is the one that is passed to the *Column.CustomPaint* event handler. Classes that implement the *ItemPaintSite* and *CustomPaintSite* interfaces can provide item painters with information about the host container in which the painting occurs.

The *CustomPaintSite* interface has the methods for retrieving and setting display-related properties, such as the background color, the foreground color, the font, the alignment setting, the margins for the item being painted, and the item's transparent state.

CustomPaintSite properties

Property	Implemented in
alignment	this class
background	this class
font	this class
foreground	this class
itemMargins	this class
siteComponent*	this class
transparent*	this class

alignment

```
public int getAlignment()
public void setAlignment(int alignment)
```

The *alignment* setting for the item being painted.

See also *com.borland.dx.text.Alignment* for alignment settings.

alignment An int representing the alignment bitmask.

background

```
public Color getBackground()
public void setBackground(Color color)
```

The background *color* for the item being painted.

color A *java.awt.Color* object representing the background color.

font

```
public Font getFont()
public void setFont(Font font)
```

The *font* to use for the item being painted.

font A *java.awt.Font* object representing the font to use.

foreground

```
public Color getForeground()
public void setForeground(Color color)
```

The foreground *color* for the item being painted.

color A *java.awt.Color* object representing the foreground color.

itemMargins

```
public Insets getItemMargins()
public void setItemMargins(Insets margins)
```

The item *margins* for the item being painted.

margins An *Insets* object representing the margins for this item.

siteComponent

```
public Component getSiteComponent()
```

Returns the hosting-site component representing the *ItemPaintSite*. This is used for coordinate space calculations, as well as to provide a component for *ItemPainter* implementations that require one, like *ImageItemPainter*, which requires an *ImageObserver* object.

transparent

```
public boolean isTransparent()
```

Whether or not the *ItemPainter* should erase its background. Returns **true** if transparent, **false** if not.

CustomPaintSite methods

Method	Implemented in
reset()	this class

reset()

```
public void reset()
```

The reset method reassigns all set values back to the defaults provided by the original *ItemPaintSite*.

DataChangeAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.DataChangeListener, java.util.EventListener

This is an adapter class for *DataChangeListener*, which is used as a notification that changes to the data in a *DataSet* have occurred.

DataChangeAdapter properties

Property	Implemented in
class*	java.lang.Object

DataChangeAdapter methods

Method	Implemented in
clone()	java.lang.Object
dataChanged(com.borland.dx.dataset.DataChangeEvent)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
postRow(com.borland.dx.dataset.DataChangeEvent)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

DataChangeEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements com.borland.jb.util.ExceptionDispatch, java.io.Serializable

The *DataChangeEvent* is the internal event generated when the data in a *DataSet* is changed. It is passed to *DataSet* components and listeners of the *DataSet*. The event ID (see the *ID* property) indicates the type of data update. Other members provide additional information on the change of the data.

The *DataChangeEvent* class may be useful for component writers, however, is not recommended for general usage.

The *DataChangeListener* responds to the *DataChangeEvent* class.

DataChangeEvent variables

Variable	Defined in
DATA_CHANGED	this class
POST_ROW	this class
ROW_ADDED	this class
ROW_CHANGE_POSTED	this class
ROW_CHANGED	this class
ROW_DELETED	this class
source	java.util.EventObject

DATA_CHANGED

```
public static final int DATA_CHANGED = 5
```

More than one row of data has changed.

POST_ROW

```
public static final int POST_ROW = 6
```

Notification to listeners that a row is posting. This allows a listener to post unposted field values just before the row is going to be posted.

ROW_ADDED

```
public static final int ROW_ADDED = 1
```

A row was added. Use *getRowAffected()* to get the new row position.

ROW_CHANGE_POSTED

```
public static final int ROW_CHANGE_POSTED = 4
```

A row was changed and posted. Use *getRowAffected()* to get the new row position.

ROW_CHANGED

```
public static final int ROW_CHANGED = 3
```

Only a cell changed, row did not post. Use *getRowAffected()* to get the new row position.

ROW_DELETED

public static final int ROW_DELETED = 2

A row was deleted. Row member has new row position. Use *getRowAffected()* to get the new row position.

DataChangeEvent constructors

DataChangeEvent(java.lang.Object, int)

public DataChangeEvent(Object source, int id)

Constructs a *DataChangeEvent* object.

DataChangeEvent(java.lang.Object, int, int)

public DataChangeEvent(Object source, int id, int affectedRow)

Constructs a *DataChangeEvent* object.

DataChangeEvent properties

Property	Implemented in
class*	java.lang.Object
exceptionChain*	com.borland.jb.util.DispatchableEvent
ID*	this class
rowAffected*	this class
source*	java.util.EventObject

ID

public final int getID()

Read-only property that indicates the type of data change. Return values for this property are constants defined in this class.

rowAffected

public final int getRowAffected()

Read-only property that returns the row affected by the data change. If the *multiRowChange* property is **false**, this property returns the row affected. Otherwise, it returns -1.

DataChangeEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
exceptionDispatch(java.util.EventListener)	this class
finalize()	java.lang.Object
hashCode()	java.lang.Object
multiRowChange()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

This method is used internally by other *com.borland* classes. You should never use this method directly.

Overrides com.borland.jb.util.DispatchableEvent.dispatch(java.util.EventListener)

multiRowChange()

public final boolean multiRowChange()

Specifies whether data in more than one row is affected. Useful for repaint strategies.

toString()

public String toString()

Returns the concatenation of *super.toString* and the value of the *ID* property.

Overrides com.borland.jb.util.DispatchableEvent.toString()

DataChangeListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.DataChangeAdapter

This interface is used to notify listeners that data has been changed or a row has been posted. Not for general usage, this interface is useful for component writers.

Developers of database applications should use the *ColumnChangeListener* for field-level validation and *EditListener* for row-level validation instead. These listeners give you a finer distinction among types of changes to data, access to data values, and the ability to block actions by throwing *VetoExceptions*.

DataChangeListener methods

Method	Implemented in
dataChanged(com.borland.dx.dataset.DataChangeEvent)	this class
postRow(com.borland.dx.dataset.DataChangeEvent)	this class

dataChanged(com.borland.dx.dataset.DataChangeEvent)

public void dataChanged(DataChangeEvent event)

An event to warn listeners that an arbitrary data change occurred to one or more rows of data.

event An object telling what type of change was made, and to which row.

postRow(com.borland.dx.dataset.DataChangeEvent)

public void postRow(DataChangeEvent event)

An event to warn listeners that a row's data has changed and must be posted.

event An object telling what type of change was made, and to which row.

DataFile class (abstract)

dx.dataset package

Extends java.lang.Object

Extended by com.borland.dx.dataset.TextDataFile

Implements com.borland.dx.dataset.Designable, java.io.Serializable

This class collects the basic behavior of all file-based data sources: loading data from and writing data to a file. These operations are often referred to as *importing* and *exporting*.

Extend this base class when creating new classes to define a custom file format that you want to import data from, or export data to.

The *TextDataFile* component extends this class. It provides the ability to read data from a text file into the *TableDataSet* component, and to save data stored in any *StorageDataSet* class object to a text file. Its properties specify how the data is organized in the text file.

DataFile properties

Property	Implemented in
class*	java.lang.Object
loadOnOpen*	this class

loadOnOpen

public abstract boolean isLoadOnOpen()

If **true**, the *StorageDataSet* will automatically load from the *DataFile* when the *StorageDataSet* is opened.

DataFile methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
load(com.borland.dx.dataset.DataSet)	this class
loadMetaData(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object

Method	Implemented in
notifyAll()	java.lang.Object
save(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

load(com.borland.dx.dataset.DataSet)

public abstract void load(DataSet dataSet)

Loads data into the *DataSet*. Implementations of this method do not need to synchronize on the *dataSet* parameter if the asynchronous *StorageDataSet* methods of *startLoading()*, *loadRow()*, and *endLoading()* are called.

loadMetaData(com.borland.dx.dataset.DataSet)

public abstract void loadMetaData(DataSet dataSet)

Implementor should load information and determine the columns of the *DataFile*.

save(com.borland.dx.dataset.DataSet)

public abstract void save(DataSet dataSet)

Saves the data in the *DataSet*.

DataFileFormat class

dx.dataset package

Extends java.lang.Object

This class defines localization variables related to data storage. Its variables state whether:

- the data is stored as 8-bit ASCII characters
- conversions from locale-specific Unicode to multibyte character sets need to take place when reading and writing data

The *DataFileFormat* is the default of the *TextDataFile* component's *fileFormat* property.

DataFileFormat variables

Variable	Defined in
ASCII	this class
ENCODED	this class

ASCII

public static final int ASCII = 1

Data is read and written as 8-bit ASCII values.

ENCODED

public static final int ENCODED = 2

Data is read and written using locale-specific Unicode to multi-byte conversions.

DataFileFormat properties

Property	Implemented in
class*	java.lang.Object

DataFileFormat methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

DataModule interface

dx.dataset package

Extends com.borland.dx.dataset.Designable

The *DataModule* is an interface that you implement when creating your custom data module class. Data modules (often referred to as *data models*) are specialized containers where data access components and their associated properties are collected into a reusable component. You define your data module once, then use it among various applications, or various frames within a single application.

The data module also provides a centralized location where “business logic” can be stored. “Business logic” describes the rules by which data is manipulated before and after the user (or client) sees the data. Business logic may include default values, filter criterion, constraints on data insertions, and so on. By encapsulating such logic in a single location, you are assured that every application that uses your data module provides consistent business logic.

The *DataModule* interface does not contain any variables, properties, methods, or events, however, by implementing this interface, you implicitly declare that your class is a data module.

When referencing your data module in your application, you can access a single instance of your data module shared across your application instead of allocating memory for multiple instances. To do this, your custom data module class should define a **public static** method that retrieves the current instantiation of the data module. This same method should also create and return a new instantiation if one doesn’t already exist. For an example of this code, use the Data Module wizard to create a data module and examine the logic of the code it generates.

Tip When working with *DataModules*, you must compile your *DataModule* class before it can be referenced in your project.

DataRow class

dx.dataset package

Extends com.borland.dx.dataset.ReadWriteRow

Implements java.io.Serializable

The *DataRow* contains one row’s worth of storage for *Column* components of the *DataSet* it is constructed from. It is useful for adding, updating, and locating rows in a *DataSet*.

A *DataRow* must be created with the *DataSet* it is used with. If the structure of a *DataSet* changes, old *DataRow* objects will not work with the updated *DataSet* and a new *DataRow* must be created for it.

When using a *DataRow* to locate data, all columns in the *DataRow* are included in the locate operation. To limit the locate to include only specified columns, use a “scoped” *DataRow*. A scoped *DataRow* includes only specified columns and is created using the *DataRow(DataSet, String)* or the *DataRow(DataSet, String[])* constructor.

To write code that handles columns of any data type, use the *setVariant(String, Variant)* or the *setVariant(int, Variant)* method and the *getVariant(String, Variant)* or *getVariant(int, Variant)* method. For example, use these methods when writing code for locating data that is not data type dependent.

Setting values in a *DataRow* does not automatically perform *Column* level constraint checks such as minimum value, maximum value or *readOnly*. This allows you to use the *DataRow*, for example, when locating a value in a calculated (*readOnly*) column. To explicitly apply constraint tests on all columns, call the *DataRow.validate()* method.

When working with *DataRows* and calling superclass methods such as *ReadRow.getBigDecimal(int)* which refer to a *Column* ordinal (*int*), the ordinal is the position of the *Column* in the *DataRow* and not of the original *DataSet*. In addition, *Column* numbering begins with zero (0) as the first *Column*. To avoid hard-coding the mapping between the *DataSet Column* order and scoped *Column* order, use the following syntax:

```
dataRow1.setString(dataset.getColumn(ordinal).getColumnName(), "test")
```

For most methods taking an ordinal parameter there is an equivalent one taking a *String* parameter. The latter is preferred over the ordinal one since column names are more readable and are not affected by changes in column order.

DataRow constructors

DataRow(com.borland.dx.dataset.DataSet)

```
public DataRow(DataSet dataSet)
```

Constructs a *DataRow* containing all the *Column* components of the specified *DataSet*, but no data values. On error, this constructor throws a *DataSetException*.

dataSet

The *DataSet* component from which to clone the structure for the *DataRow*. All *Column* components of the *DataSet* are included in the *DataRow*.

DataRow(*com.borland.dx.dataset.DataSet*, *java.lang.String*)

public DataRow(DataSet dataSet, String columnName)

Creates a “scoped” *DataRow* containing the structure of (but no data from) the specified column of the current row position. On error, this constructor throws a *DataSetException*.

<i>dataSet</i>	The <i>DataSet</i> component from which to clone the structure for the <i>DataRow</i> . Only the <i>Column</i> component specified in the <i>columnName</i> parameter is included in the <i>DataRow</i> .
<i>columnName</i>	The <i>String</i> name of the <i>Column</i> to include in the <i>DataRow</i> .

DataRow(*com.borland.dx.dataset.DataSet*, *java.lang.String*[])

public DataRow(DataSet dataSet, String[] columnNames)

Constructs a “scoped” *DataRow* containing the data structure (but no data) from specified columns of the *DataSet*. On error, this constructor throws a *DataSetException*.

<i>dataSet</i>	The <i>DataSet</i> component from which to clone the structure for the <i>DataRow</i> . All <i>Column</i> components of the <i>DataSet</i> are included in the <i>DataRow</i> .
<i>columnNames</i>	An array of <i>String</i> names of the <i>Column</i> components to include in the <i>DataRow</i> .

DataRow properties

Property	Implemented in
assignedNull**	com.borland.dx.dataset.ReadWriteRow
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns*	com.borland.dx.dataset.ReadRow
unassignedNull**	com.borland.dx.dataset.ReadWriteRow

DataRow methods

Method	Implemented in
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow

Method	Implemented in
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hashCode()	java.lang.Object
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.ReadRow
isModified(java.lang.String)	com.borland.dx.dataset.ReadRow
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
notify()	java.lang.Object
notifyAll()	java.lang.Object
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.ReadWriteRow
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
toString()	com.borland.dx.dataset.ReadRow
validate()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

validate()

```
public final void validate()
```

Tests all columns in the *DataRow* for constraints on the data such as minimum or maximum value, readOnly, and so on.

DataSet class (abstract)

dx.dataset package

Extends	com.borland.dx.dataset.ReadWriteRow
Extended by	com.borland.dx.dataset.DataSetView, com.borland.dx.dataset.StorageDataSet
Implements	com.borland.dx.dataset.AccessListener, com.borland.dx.dataset.Designable, com.borland.dx.dataset.MasterNavigateListener, com.borland.dx.dataset.StatusListener, java.io.Serializable, java.util.EventListener

The *DataSet* class is an abstract class that provides basic editing, view, and cursor functionality for access to two-dimensional data. It supports the concept of a current row position which allows for navigation of the data in the *DataSet*. The *DataSet* also manages a *pseudo* record—an area in memory where a newly inserted row or changes to the current row are temporarily stored.

The data in a *DataSet* can be modified programmatically or through a data-aware control. To connect a *DataSet* to a data-aware control, set the control's *dataSet* property to the *DataSet* you want to use as the data source. Several data-aware controls can be associated with the same *DataSet*. In such cases, the controls navigate together and when you move the row position of a control, the row position changes for all controls that share the same *DataSet*. This synchronization of controls that share a common *DataSet* can greatly ease the development of the user-interface portion of your application.

Controls which share the same *DataSet* as their data source share also the same pseudo record. This allows updates to be visible as soon as entry at the field level is complete, for example, by navigating off the field.

The *DataSet* component is opened implicitly (by default) when visual components bound to it are shown. For example, launching an application that includes a data-aware control that is bound to a *DataSet* automatically opens the *DataSet* so you seldom have to open the *DataSet* explicitly. No code is generated for this implicit open.

You can navigate through the data in a *DataSet* using a UI control or programmatically. To navigate programmatically, use the *first()*, *last()*, *next()*, and *prior()* methods. To verify if such a navigation is valid use the *inBounds()* method. Similarly, to perform tests for the “end of file” or “beginning of file” conditions, use the *atLast()* or *atFirst()* methods.

The *DataSet* has an associated *SortDescriptor* object that defines properties which affect the sort order of the data in the *DataSet*.

DataSet objects have relational capability and can be linked to form master-detail relationships. The *MasterLinkDescriptor* holds the properties required for relational capability between *DataSet* objects.

The *DataSet* class is extended by *DataSetView* and *StorageDataSet*. The *DataSetView* component allows for an alternate view (sort order, navigation, and filter criteria) for the data contained in the *DataSet*. A *StorageDataSet*, and any class that extends *StorageDataSet*, manages the storage aspects of the data operated on by the *DataSet*.

DataSet properties

Property	Implemented in
assignedNull**	com.borland.dx.dataset.ReadWriteRow
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns*	com.borland.dx.dataset.ReadRow
defaultValues**	this class
detailDataSetWithFetchAsNeeded*	this class
details*	this class
displayErrors	this class
editable	this class
editing*	this class
editingNewRow*	this class
empty*	this class
enableDelete	this class
enableInsert	this class
enableUpdate	this class
internalRow*	this class
lastColumnVisited	this class
masterLink	this class
open*	this class
row*	this class
rowCount*	this class
rowFilterListener*	this class
schemaName*	this class
sort	this class
status*	this class
storageDataSet*	this class
tableName*	this class
unassignedNull**	com.borland.dx.dataset.ReadWriteRow

defaultValues

```
public void setDefaultValues(DataRow row)
```

Initializes the current row to the default values of all columns. If the *DataSet* is a detail *DataSet* of a master-detail relationship, this method also initializes the detail linking columns.

See also *setDefaultValues()* property of the *DataRow*

detailDataSetWithFetchAsNeeded

```
public final boolean isDetailDataSetWithFetchAsNeeded()
```

Read-only property that returns **true** if a *MasterLinkDescriptor* has been set and its *isFetchAsNeeded* property is **true**. Otherwise, it returns **false**.

details

```
public final synchronized DataSet[] getDetails()
```

Read-only property that returns an array of all detail *DataSets* associated with this master *DataSet*.

displayErrors

```
public final boolean isDisplayErrors()
```

```
public final void setDisplayErrors(boolean displayErrors)
```

Determines whether the *DBExceptionDialog* displays or not when an *Exception* is generated.

editable

```
public final boolean isEditable()
```

```
public final void setEditable(boolean editable)
```

If **true**, edits to this *DataSet* from a UI control are not allowed; if **false**, edits from a UI control are allowed. If an edit is attempted when this property is set to **true**, a *ValidationException* is thrown. UI controls have default error handling that display this in a *StatusControl* or *JdbStatusLabel*, if one is present. If there is no status control, the error is displayed in an error dialog. If this property is set on a *DataSetView*, only that view is affected by this setting.

See also *readOnly* property of the *StorageDataSet* class, *resolvable* property of the *StorageDataSet* class

editing

```
public final boolean isEditing()
```

Read-only property that returns whether the *DataSet* is being edited or not.

editingNewRow

public final boolean isEditingNewRow()

Read-only property that returns whether data is being added to a new row in the *DataSet* or not.

empty

public final boolean isEmpty()

Read-only property that returns whether the *DataSet* is empty or contains data.

enableDelete

public final boolean isEnabledDelete()

public final void setEnableDelete(boolean enable)

Enables/disables row deletions. If a delete is attempted, a *ValidationException* is thrown. UI controls have default error handling that display this in a *StatusControl*, if one is present. If there is no *StatusControl*, the error is displayed in an error dialog. If this property is set on a *DataSetView*, only that view is affected by this setting.

enableInsert

public final boolean isEnabledInsert()

public final void setEnableInsert(boolean enable)

Enables/disables row insertions. If a row insertion is attempted, a *ValidationException* is thrown. UI controls have default error handling that display this in a *StatusControl*, if one is present. If there is no *StatusControl*, the error is displayed in an error dialog. If this property is set on a *DataSetView*, only that view is affected by this setting.

enableUpdate

public final boolean isEnabledUpdate()

public final void setEnableUpdate(boolean enable)

Enables/disables row updates. If a row update is attempted, a *ValidationException* is thrown. UI controls have default error handling that display this in a *StatusControl*, if one is present. If there is no *StatusControl*, the error is displayed in an error dialog. If this property is set on a *DataSetView*, only that view is affected by this setting.

internalRow

```
public final long getInternalRow()
```

Returns a unique identifier for the current row. Can be used by the *goToInternalRow* method to reposition the current row position. The *internalRow* assignment for a row never changes, and is never reused by another row. Setting this property is the fastest way to navigate to an arbitrary row.

lastColumnVisited

```
public final String getLastColumnVisited()
public final void setLastColumnVisited(String columnName)
```

Specifies the most recent *Column* that was navigated to in a data-aware control. This method is used by the *interactiveLocate()* method as the default *Column* to locate on if its parameter of a target *Column* for the locate is not specified.

masterLink

```
public MasterLinkDescriptor getMasterLink()
public synchronized void setMasterLink(MasterLinkDescriptor descriptor)
```

Stores the *MasterLinkDescriptor* object that contains the properties of a master-detail relationship. This property is set for the detail *DataSet* of a master-detail relationship and is applicable for any *StorageDataSet* implementation, for example, *QueryDataSet*. This property can be used with a *DataSetView* component only if the *fetchAsNeeded* property of the *MasterLinkDescriptor* if **false**.

descriptor The *masterLink* property.

open

```
public boolean isOpen()
```

Read-only property that returns whether the *DataSet* is open or not.

row

```
public final int getRow()
```

Returns current row position.

rowCount

```
public final int getRowCount()
```

Returns the number of rows visible with this *DataSet*. This number may be different than the number of rows in the associated *StorageDataSet* if a *RowFilterListener* is being used to filter out some rows of the *DataSet*.

Also, when a *Provider*, such as the *QueryProvider* used by *QueryDataSet*, is loading rows asynchronously, attempting to access the *DataSet* data or *rowCount* property may lead to unexpected results. This is due to the way the rows are being loaded. The *rowCount* will continue to increase as 0 or more rows are loaded, until the asynchronous load operation is completed.

rowFilterListener

```
public final RowFilterListener getRowFilterListener()
```

Read-only property that returns the registered *RowFilterListener* of the *DataSet*.

schemaName

```
public String getSchemaName()
```

Read-only property that returns the schema name.

sort

```
public final SortDescriptor getSort()
public final void setSort(SortDescriptor descriptor)
```

Specifies the *SortDescriptor* object where properties that define the sort order of the data in a *DataSet* are stored.

Setting the sort descriptor generates a dispatch to the registered *AccessListener* object. On error, *setSort()* generates a *DataSetException*.

status

```
public final int getStatus()
```

Read-only property that returns the status for the current row of the *DataSet*. Valid values for the returned *int* are defined in *com.borland.dx.dataset.RowStatus* variables.

```
Example    // Code checks whether the row has been newly inserted or the
           // inserted row resolved back to the data source. If so, processing code is
           // executed.
           if ( (queryDataSet1.getStatus() & (RowStatus.INSERTED) ) != 0)
           {
               // processing code here
           }
```

storageDataSet

```
public final StorageDataSet getStorageDataSet()
```

Read-only property that specifies the name of the *StorageDataSet* object that manages the storage of the rows operated on by the *DataSet*.

tableName

```
public String getTableName()
```

Read-only property that specifies the name of a table from which the *DataSet* obtains its data. This property must be set for *DataSet* objects that have updateable columns.

The *tableName* property is normally retrieved automatically from the data source. Some stored procedures and queries however will not return the table name. If it doesn't, use the *setTableName()* property of *StorageDataSet* to specify the table so that changes made to the *DataSet* can be resolved back to the data source. The table name is also useful as a *String* identifier of the data source.

DataSet methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	this class
addRow(com.borland.dx.dataset.DataRow)	this class
allocateValues()	this class
atFirst()	this class
atLast()	this class
cancel()	this class
cancelLoading()	this class
cancelOperation()	this class
canNavigate (com.borland.dx.dataset.Column, int)	this class
canSet(com.borland.dx.dataset.Column)	this class
clearStatus()	this class
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneDataSetView()	this class
close()	this class
columnIsVisible(java.lang.String)	this class
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow

Method	Implemented in
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	this class
deleteRow()	this class
dittoRow(boolean, boolean)	this class
dittoRow(boolean)	this class
dropIndex()	this class
editRow()	this class
emptyAllRows()	this class
emptyRow()	this class
enableDataSetEvents(boolean)	this class
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	this class
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDataRow (com.borland.dx.dataset.DataRow)	this class
getDataRow(int, com.borland.dx.dataset.DataRow)	this class
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getDetail(java.lang.String)	this class
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	this class
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	this class
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	this class
goToClosestRow(int)	this class
goToInternalRow(long)	this class
goToRow(com.borland.dx.dataset.ReadRow)	this class
goToRow(int)	this class
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	this class
hashCode()	java.lang.Object
hasValidations()	this class
inBounds()	this class
insertRow(boolean)	this class

Method	Implemented in
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	this class
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	this class
isModified(java.lang.String)	this class
isNew(int)	this class
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	this class
locate(com.borland.dx.dataset.ReadRow, int)	this class
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	this class
masterNavigated(com.borland.dx.dataset.MasterNavigateEvent)	this class
masterNavigating(com.borland.dx.dataset.MasterNavigateEvent)	this class
moveRow(int)	this class
next()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	this class
openDetails()	this class
post()	this class
postAllDataSets()	this class
prior()	this class
refetchRow (com.borland.dx.dataset.ReadWriteRow)	this class
refilter()	this class
refresh()	this class
refreshSupported()	this class
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
resetInBounds()	this class
resetPendingStatus(boolean)	this class
resetPendingStatus(long, boolean)	this class
saveChanges()	this class
saveChangesSupported()	this class
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	this class
setDisplayVariant(int, com.borland.dx.dataset.Variant)	this class
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	this class
startEditCheck (com.borland.dx.dataset.Column)	this class
statusMessage (com.borland.dx.dataset.StatusEvent)	this class
statusMessage(int, java.lang.String)	this class
toggleViewOrder(java.lang.String)	this class
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	this class
validate()	this class
validate(com.borland.dx.dataset.ReadRow)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

addRow(com.borland.dx.dataset.DataRow)

```
public final void addRow(DataRow dataRow)
```

Adds a new row to the *DataSet* at the end of existing rows and sets the new row's *RowStatus* to *INSERTED*. The new row is validated against data constraints (if any) to ensure that only valid data is added.

This method adds a new, unposted “pseudo” row into the *DataSet*. Until *post()* is called, this row will not be visible to *DataSetView* components that are sharing the same *StorageDataSet*. After *addRow()* has been called, values can be set in the pseudo row by calling *DataSet* set methods.

When the *post()* method is called, the unposted row is stored in the *StorageDataSet* and becomes visible by *DataSetViews* sharing the same *StorageDataSet*. When the row is posted, and the *sort* property is not set, the current row in the *DataSet* will change to the newly-posted row at the end of the *DataSet*. If the *DataSet* *sort* property is set, the current row of the *DataSet* will be the newly-posted row, displayed in its sorted position within the *DataSet*.

Many navigating methods will cause an unposted row to be automatically posted. In this case, the new row appears to “fly away” to its proper position, which may be at the end of current rows if the *sort* property is **null**, or to its sorted position if the *sort* property is not **null**. In these cases, the current row is not necessarily (and is not likely to be) the newly-posted row. Instead the cursor is positioned according to the navigating request. For example, if the *prior()* method is called while an unposted row exists, the new row will be

posted into the *StorageDataSet* according to the *sort* property setting, but the current row position will be one row up.

dataRow The *DataRow* to add to the current *DataSet*.

allocateValues()

```
public final Variant[] allocateValues()
```

Allocates an array of *Variants* for all the *Columns* in the *DataSet*.

atFirst()

```
public final boolean atFirst()
```

Returns **true** if the current position is the first visible row of *DataSet*, otherwise, this method returns **false**. This method is the equivalent of the *BOF()* (beginning of file) function that is available in many programming languages.

atLast()

```
public final boolean atLast()
```

Returns **true** if the current position is the last visible row of *DataSet*, otherwise, this method returns **false**. This method is the equivalent of the *EOF()* (end-of-file) function available in many programming languages.

cancel()

```
public final void cancel()
```

Cancels the edits to a new or existing record. On error, this method throws a *DataSetException*.

cancelLoading()

```
public void cancelLoading()
```

Cancels the loading of data into the *DataSet*. This method can be used to terminate a long running asynchronous provider operation. The *QueryProvider* used by *QueryDataSet* has *Load* options that allow it to load query results into a *StorageDataSet* asynchronously on a separate thread. Calling this method will cause such an operation to terminate.

See also *cancelOperation()* method

cancelOperation()

public void cancelOperation()

Cancels any long running operation currently active on this *DataSet*. This includes loading a query result, restructure operations, and index building operations. For this have an effect, the long running operation must be executing on a different thread than the thread that calls this method.

canNavigate(com.borland.dx.dataset.Column, int)

public final boolean canNavigate(Column column, int row)

Returns whether the specified *column* can be navigated to (**true**) or not (**false**). This method returns **false** if a *Column*'s *readOnly* property is set to **true**. This method is typically used by visual components.

column The *Column* component to navigate to.

row The row position in the *DataSet*.

canSet(com.borland.dx.dataset.Column)

public final boolean canSet(Column column)

Returns whether the values at the current row and specified *Column* can be edited.

clearStatus()

public final void clearStatus()

Instructs all status listeners (for example, the *StatusBar* control) to clear their status messages.

cloneDataSetView()

public final DataSetView cloneDataSetView()

Copies the *DataSetView* and returns the copied object. On error, this method throws a *DataSetException*.

close()

public boolean close()

This method returns **true** if the *DataSet* is closed by executing this method and **false** if the *DataSet* does not need closing (for example, it is already closed).

Closing a *DataSet* allows for structural changes to be made to the *DataSet*, such as adding a new *Column* or changing the sort order, and so on. All

changes are applied to the data currently in the *DataSet*. The *close()* method does not discard any inserted, edited or deleted rows in the *DataSet*.

For the *DataSetView* subclass of *DataSet*, the *close()* method must be called when explicitly removing an instantiated *DataSetView* be to garbage-collected. Otherwise, the instantiated *DataSetView* remains allocated in memory as long as its associated *StorageDataSet* cannot be garbage-collected.

If you connect to your data source using a *QueryDataSet* and its query statement contains parameters, you need to call *DataSet.close()* before providing for another *QueryDataSet*.

columnIsVisible(java.lang.String)

```
public boolean columnIsVisible(String columnName)
```

Indicates **true** if the *Column* should be displayed in UI controls. This method returns **true** for *Columns* that have their *visible* property set. Aggregate calculated columns and master-detail link columns are hidden by default and return **false**.

columnName The *String* name of the *Column* component.

deleteAllRows()

```
public final void deleteAllRows()
```

Deletes all rows of data visible in the *DataSet*. If called on a detail *DataSet*, only the rows in the current group are deleted. Deletion status information is tracked for resolution back to the original data source. If all rows of the *DataSet* cannot be deleted, this method throws a *DataSetException*.

See also *emptyAllRows()* to remove visible rows in the *DataSet* without having them marked as deleted

deleteRow()

```
public final void deleteRow()
```

Deletes the current row of the *DataSet*. If the current row is new or being edited, the edit state is canceled and the new or edited row is deleted. On failure, this method throws a *DataSetException*.

dittoRow(boolean)

```
public void dittoRow(boolean dittoExisting)
```

Copies the contents of the previous row to the current (new) row. Does nothing if the current row is the first row. If the current row is an existing row, and you want to copy the contents of the previous row to it, set

dittoExisting to **true**. By default, *dittoExisting* is **false**, as copying over the contents of an existing row could cause data loss.

dittoExisting If **true**, copy the contents of the previous row to an existing row. By default this option is **false**, as it could cause data loss.

dittoRow(boolean, boolean)

public void dittoRow(boolean dittoExisting, boolean autoInsert)

Copies the contents of the previous row to the current row. Does nothing if the current row is the first row.

dittoExisting If **true**, copies the contents of the previous row to an existing row. By default this option is **false**, as it could cause data loss.

autoInsert Automatically inserts a new row if the *DataSet* is not in edit mode, then copies the contents of the previous row to the new row.

dropIndex()

public synchronized boolean dropIndex()

Deletes the index (if applicable) that maintains the sort order created according to the *sort* property and the *rowFilter* event of the *DataSet*.

Note If the *DataSet* is immediately reopened without changing the *sort* property or *rowFilter* event listener of this class, the index will be rebuilt.

editRow()

public final void editRow()

Edits the existing row of the *DataSet*. When the edits are posted, the row's status flag is set to *UPDATED*. On failure, this method throws a *DataSetException*.

emptyAllRows()

public final void emptyAllRows()

This method calls the *emptyRow()* method for all rows visible in the *DataSet*. If called on a detail *DataSet*, only the rows in the current group are emptied.

The emptied rows are not marked as deleted rows. However, they are completely forgotten and cannot be resolved back to a data source.

See also *deleteAllRows()*

emptyRow()

```
public final void emptyRow()
```

Empty the current *DataSet* row. The emptied row is not marked as deleted. However, it is completely forgotten and cannot be resolved back to a data source.

See also *deleteRow()*

enableDataSetEvents(boolean)

```
public void enableDataSetEvents(boolean enable)
```

Affects whether data-aware controls ignore changes or repaint as a result of programmatic changes to the position and general state of a *DataSet*. A control cannot repaint properly while the *DataSet events* are disabled.

When performing lengthy *DataSet* operations, you may want to set this property to **false** prior to initiating such operations. If you do, you must set this property back to **true** to allow UI controls to repaint properly. You may also want to execute this method to propagate changes made to a *DataSet* object's structure to a control that is bound to the *DataSet*.

first()

```
public final void first()
```

Moves the row position to the first row visible to the *DataSet* and sends a *NavigationEvent* to registered *NavigationListeners* if the move was successful. This may cause the *inBounds()* method to return **true** if more than one row exists.

getDataRow(com.borland.dx.dataset.DataRow)

```
public final void getDataRow(DataRow dataRow)
```

Returns all the values at the current row.

dataRow The *DataRow* object that gets its values from the current row position when this method is executed.

getDataRow(int, com.borland.dx.dataset.DataRow)

```
public final void getDataRow(int row, DataRow dataRow)
```

Returns all the values at *row*.

row The unique *rowId* for a row in the *DataSet*.

dataRow The *DataRow* object that gets its values from the row indicated by the *row* parameter when this method is executed.

getDetail(java.lang.String)

public final synchronized DataSet getDetail(String tableName)

Returns the detail *DataSet* with given *tableName* property setting. This method throws a *DataSetException.UNKNOWN_DETAIL_NAME* if there is no detail with *tableName*. It is possible (but not likely) that more than one detail has the same *tableName*.

getDisplayVariant(int, int, com.borland.dx.dataset.Variant)

public final void getDisplayVariant(int ordinal, int row, Variant value)

Called by data-aware controls. Sets the *value* parameter to the value that should be displayed in the visual control. Normally this is the same as what *getVariant(int, int, com.borland.dx.dataset.Variant)* returns. If the column at the ordinal position has its *pickList* property set, the *value* parameter may be initialized to the value from another column in a separate pick list *DataSet*. This provides the capability to store a compact “code” value, but display a more “descriptive” value. This behavior is enabled when the column at position *ordinal* has its *calcType* property set to *LOOKUP* and it has a defined *PickListDescriptor* with its *lookupDisplayColumn* property set to a column name in the pick list *DataSet*. On error, this method throws a *DataSetException*.

getVariant(int, int, com.borland.dx.dataset.Variant)

public final void getVariant(int ordinal, int row, Variant value)

Assigns the value stored at the intersection of the specified *Column* and *row* to *value*.

<i>ordinal</i>	An integer value that represents the <i>n</i> th <i>Column</i> in the <i>DataSet</i> where the <i>Variant</i> value is located.
<i>row</i>	A integer value representing the unique row identifier for the row.
<i>value</i>	The value that is assigned by executing this method.

See also *getDisplayVariant(int, int, com.borland.dx.dataset.Variant)*

getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)

public final void getVariant(String columnName, int row, Variant value)

Assigns the value stored at the intersection of the specified *columnName* and *row* to *value*.

<i>columnName</i>	The <i>String</i> name of the <i>Column</i> .
<i>row</i>	The unique row identifier for the row.
<i>value</i>	The value that is assigned by executing this method.

goToClosestRow(int)

```
public final boolean goToClosestRow(int row)
```

Moves the row position to the closest row indicated by the *row* parameter. This method returns **true** if the move is successful, otherwise, it returns **false**.

row The unique row identifier for the row of the *DataSet* that this method should attempt to move to.

On error, this method throws a *DataSetException*.

goToInternalRow(long)

```
public final boolean goToInternalRow(long newInternalRow)
```

Attempts to navigate to *newInternalRow*. If *newInternalRow* no longer exists, position remains unchanged. Returns **true** for success.

goToRow(com.borland.dx.dataset.ReadRow)

```
public void goToRow(ReadRow row)
```

Forces this *DataSet* to move to the same position as *row* if *row* is of the same *StorageDataSet* as this *DataSet*. If *row* is not from the same *StorageDataSet*, this method throws a *DataSetException* of *INCOMPATIBLE_DATA_ROW*. If this *DataSet* object's current view is filtered and does not include *row*, the current position of this *DataSet* moves to the closest row that is in its view.

row The unique row identifier of the row to move to.

goToRow(int)

```
public final boolean goToRow(int row)
```

Moves to the specified row position, where *row* represents the unique row identifier for the row. Returns **true** if the move is successful, otherwise, it returns **false**. On error, this method throws a *DataSetException*.

row The unique row identifier of the row to move to.

hasDetail(java.lang.String)

```
public final synchronized DataSet hasDetail(String tableName)
```

Returns the detail *DataSet* with given *tableName* property setting or **null** if there is no detail with *tableName*.

Note It is possible, but not likely, that more than one detail has the same *tableName*.

hasValidations()

public final boolean hasValidations()

Returns **true** if validation constraints are in affect for this row.

inBounds()

public final boolean inBounds()

Returns **true** if the most recent navigation was *in bounds*, otherwise, this method returns **false**. A navigation is in bounds if it falls between the first and last records (inclusive) that are visible to the cursor. This method allows you to test whether either condition of beginning or end of file is encountered with a single method call.

Methods that set the “in bounds” flag include

- *first()*
- *last()*
- *next()*
- *prior()*

If you prefer or if your application requires it, you can call any of these methods to reset the *inBounds* flag. Any navigation effected through the user-interface also sets *inBounds()* flag.

insertRow(boolean)

public final void insertRow(boolean before)

Start editing a new row. The *before* parameter indicates whether to insert before or after the current row.

This method adds a new, unposted “pseudo” row into the *DataSet*. Until *post()* is called, this row will not be visible to *DataSetView* components that are sharing the same *StorageDataSet*. After *insertRow()* has been called, values can be set in the pseudo row by calling *DataSet* set methods. The *insertRow()* method provides the functionality of row inserting in data aware visual components.

When the *post()* method is called, the unposted row is stored in the *StorageDataSet* and becomes visible by *DataSetViews* sharing the same *StorageDataSet*. When the row is posted, and the *sort* property is not set, the current row in the *DataSet* will change to the newly-posted row at the end of the *DataSet*. If the *DataSet* *sort* property is set, the current row of the *DataSet* will be the newly-posted row, displayed in its sorted position within the *DataSet*.

Many navigating methods will cause an unposted row to be automatically posted. In this case, the new row appears to “fly away” to its proper position, which may be at the end of current rows if the *sort* property is **null**, or to its sorted position if the *sort* property is not **null**. In these cases, the current row is not necessarily (and is not likely to be) the newly-posted row. Instead the cursor is positioned according to the navigating request. For example, if the *prior()* method is called while an unposted row exists, the new row will be

posted into the *StorageDataSet* according to the *sort* property setting, but the current row position will be one row up.

See also *addRow()* property for higher performance batch row adding

before Whether the new row should be inserted before (**true**), or after (**false**), the current row. This property defaults to **false**.

interactiveLocate(java.lang.String, java.lang.String, int, boolean)

```
public void interactiveLocate(String text, String locateColumnName, int locateOptions, boolean enterPressed)
```

Searches the specified *Column* of the *DataSet* for the value specified in *text*.

<i>text</i>	The <i>String</i> representation of the value to search for.
<i>locateColumnName</i>	The <i>String</i> name of the <i>Column</i> in which to perform the search. If null , this method attempts to use the last column visited (as returned by <i>getLastColumnVisited()</i>). If the last column visited was not set, the first <i>Column</i> in the <i>StorageDataSet</i> is used.
<i>locateOptions</i>	An integer value representing the locate options to use. See <i>com.borland.dx.dataset.Locate</i> for valid values for this parameter.
<i>enterPressed</i>	Specifies whether the <i>Enter</i> key should be pressed prior to initiating the search or if each key press should be used to perform an incremental search. Incremental searches are available only for <i>String</i> searches.

isModified(int)

```
public final boolean isModified(int ordinal)
```

Returns **true** if the value at the specified *ordinal* has been modified.

Overrides *com.borland.dx.dataset.ReadRow.isModified(int)*

isModified(java.lang.String)

```
public final boolean isModified(String columnName)
```

Returns **true** if the value at the specified *columnName* has been modified.

Overrides *com.borland.dx.dataset.ReadRow.isModified(java.lang.String)*

isNew(int)

public final boolean isNew(int row)

Returns whether the row indicated by the *row* parameter is new (**true**). Otherwise, this method returns **false**.

row The *row* parameter represents the unique row identifier for the row.

On failure, this method throws a *DataSetException*.

last()

public final void last()

Moves the current position to the last row visible to the *DataSet* and sends a *NavigationEvent* notification to registered *NavigationListeners* if the move was successful. This may cause the *inBounds()* to return **true** if more than one row exists.

If an edit is in progress, the changes are posted prior to performing the move operation. On failure, this method throws a *DataSetException*.

locate(com.borland.dx.dataset.ReadRow, int)

public final boolean locate(ReadRow rowLocate, int locateOptions)

Locates the row of data with the specified row of values and moves the current row position to that row. The locate operation includes all columns of the *ReadRow*; to limit the locate to specific columns of interest, use a *scoped DataRow*. If the row is scoped to a specific set of columns, only those columns are used for the locate.

rowLocate The *ReadRow* that contains values to use in for the locate operation.

locateOptions Options that are applicable when locating data. Valid values for the *locateOptions* are defined in *com.borland.dx.dataset.Locate* variables. The *Locate* variables may be combined using the Java bit-wise OR operator of a vertical pipe symbol (|) between each variable where it makes sense to do so. For example, you can search using partial strings and specifying case insensitivity.

lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)

public final boolean lookup(ReadRow locateRow, DataRow resultRow, int locateOptions)

Performs a lookup for the row with the specified values. If a match is found, this method returns the data from the matching row as *resultRow*, and

returns **true** but does not navigate to the matching row. If no match is found, this method returns **false**. This method includes all columns of the *ReadRow*; to limit the lookup to specific columns, use a “scoped” *DataRow* that includes only the columns of interest.

<i>locateRow</i>	The <i>ReadRow</i> that contains values to use in for the lookup operation.
<i>resultRow</i>	The data in the row where the match with the <i>rowLocate</i> values is found.
<i>locateOptions</i>	Options that are applicable when locating data. Valid values for the <i>locateOptions</i> are defined in <i>com.borland.dx.dataset.Locate</i> variables. The <i>Locate</i> variables may be combined using a vertical pipe symbol () between each variable where it makes sense to do so. For example, you can search using partial strings and specifying case insensitivity.

moveRow(int)

```
public final int moveRow(int delta)
```

For *DataSets* that have the *Sort.SortAsInserted* property set to **true**, this method can be used to move the row.

next()

```
public final boolean next()
```

Moves the row position to the next row visible to the *DataSet* and sends a *NavigationEvent* notification of *rowChanged* to its registered *NavigationListeners* if the move was successful. The *inBounds()* method returns **false** if *next()* is called when the *DataSet* is positioned at the last visible row.

On failure, this method throws a *DataSetException*.

open()

```
public boolean open()
```

If the *DataSet* is not open, this method attempts to open the *DataSet*. The method returns **true** if the *DataSet* can be successfully opened. If the *DataSet* is already open, the method returns **false**. If an error was encountered opening the *DataSet*, a *DataSetException* is thrown.

openDetails()

public final synchronized void openDetails()

Call this method on a master *DataSet* to force all details to open. By default, a detail will ask its master *DataSet* to open, but a master *DataSet* does not ask the detail *DataSet* to open.

post()

public final boolean post()

If a new or existing record is being edited, calling *post()* force the record to be posted. If this method is called for an unposted row, the row will be posted in the *StorageDataSet* with a *RowStatus* of INSERTED. If the *sort* property is not set, the new row will be positioned at the end of the *StorageDataSet*. If the *sort* property is set, the row will be positioned according to the *sort* property setting.

If called for an existing row, the *RowStatus* for that row is UPDATED, if it does not already have the *RowStatus* of INSERTED. As with new rows, it will be positioned according to the *sort* property setting.

postAllDataSets()

public void postAllDataSets()

Posts all unposted rows for all *DataSet* and *DataSetView* components that share the same *StorageDataSet* property. On error, this method throws a *DataSetException*.

prior()

public final boolean prior()

Moves the row position to the previous row visible to the *DataSet* and sends a *NavigationEvent* notification of *rowChanged* to its registered *NavigationListeners* if the move was successful. This will cause the *inBounds()* method to return **false** if *prior()* is called when the *DataSet* is positioned at the first visible row. On error, this method throws a *DataSetException*.

refetchRow(com.borland.dx.dataset.ReadWriteRow)

public void refetchRow(ReadWriteRow row)

Refetches the specified row if the *DataSet* has a data provider (for example, *QueryDataSet* and *ProcedureDataSet*) that supports this operation.

row

The *ReadWriteRow* implementation that you want refetched.

refilter()

public void refilter()

Forces the filter for this *DataSet* to be recomputed on all rows.

Closing and re-opening a *DataSet* with a filter will not always cause the filter code to be re-executed. Similarly, if your filter is based on the value of a global variable, changing the global variable will not change the set of rows displayed. In both cases, use the *refilter()* method to ensure that the filter is re-executed.

refresh()

public void refresh()

Refreshes the data from the *DataSet* if the data provider supports this operation (for example, *QueryDataSet* and *ProcedureDataSet*). On error, this method throws a *DataSetException*.

refreshSupported()

public boolean refreshSupported()

Returns whether a refresh operation is supported. For example, *QueryDataSet* and *ProcedureDataSet* components return **true**.

resetInBounds()

public final void resetInBounds()

Sets the *in bounds* state to **true**. This method is not normally used in an application's code since any successful navigation sets this to **true**.

resetPendingStatus(boolean)

public void resetPendingStatus(boolean resolved)

Reset the status bits of the rows marked pending during resolution.

resolved

true if changes were resolved and the changed rows should now be treated as originals in a new resolution query. **false** if changes were rolled back and the changed rows should still be treated as changed rows.

resetPendingStatus(long, boolean)

public void resetPendingStatus(long internalRow, boolean resolved)

Reset the status bits of a specific row.

<i>internalRow</i>	the row where the status bits should be reset.
<i>resolved</i>	true if changes were resolved and the changed rows should now be treated as originals in a new resolution query. false if changes were rolled back and the changed rows should still be treated as changed rows.

saveChanges()

public void saveChanges()

Saves changes made to the data in the *DataSet* back to the data source using default *QueryResolver* behavior.

saveChangesSupported()

public boolean saveChangesSupported()

Returns whether the data source supports the saving of data changes (**true**) or not (**false**). For *TableDataSet* components, this method returns **false** if the *resolver* property is not set.

setDefaultValues()

public void setDefaultValues()

Initializes the current row to default values of all columns. If the *DataSet* is a detail *DataSet* of a master-detail relationship, this method also initializes the detail linking columns.

Overrides com.borland.dx.dataset.ReadWriteRow.setDefaultValues()

setDisplayVariant(int, com.borland.dx.dataset.Variant)

public final void setDisplayVariant(int ordinal, Variant value)

Called by data aware controls. Sets the *value* parameter to the value that should be displayed in the visual control. Normally this is the same as what *getVariant(int, int, com.borland.dx.dataset.Variant)* returns.

If the column at the ordinal position has its *pickList* property set, the *value* parameter may be initialized to the value from another column in a separate pick list *DataSet*. This provides the capability to store a compact “code” value, but display a more “descriptive” value. This behavior is enabled when the column at position *ordinal* has its *calcType* property set to *LOOKUP* and it

validate(com.borland.dx.dataset.ReadRow)

```
public void validate(ReadRow readRow)
```

Applies min, max, required, precision checks. Calls the *Column.validate()* method if it is set. These checks will be made as data is edited in the *DataSet*, and when the changes are saved back to the database.

DataSet event listeners

This class is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

DataSetAware interface

dx.dataset package

Extended by com.borland.dx.dataset.ColumnAware

The *DataSetAware* interface is a way for a component to declare to JBuilder that it knows how to bind to a *DataSet*. Note that all the interface methods defined here are valid JavaBeans property getters/setters by design to simplify the component's implementation. The *ColumnAware* interface extends this to include the extra granularity of a single column setting.

The *DataSetAware* interface has one property only that must be implemented, *dataSet*.

DataSetAware properties

Property	Implemented in
dataSet	this class

dataSet

public DataSet getDataSet()
public void setDataSet(DataSet dataSet)

Determines the *DataSet* the control that implements this interface is linked to.

DataSetData class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.LoadCancel, java.io.Serializable

The *DataSetData* class allows you to separate the state of a *DataSet* (stored by its properties) from its data by extracting only data. This class is serializable, and may be used to stream the data to an output stream, or to simply be passed as an argument to a Remote Method Invocation (RMI) method. The *DataSetData* class also allows you to load the data back into another *DataSet*.

Use the *DataSetData* class when developing 3-tier database applications.

While this class separates data from its properties, a few metadata-type column properties are stored within this class. These are:

- column name
- datatype
- rowId
- precision
- scale
- whether the column is hidden

This metadata information is obtained when the *DataSetData* object is populated with data.

DataSetData properties

Property	Implemented in
class*	java.lang.Object

DataSetData methods

Method	Implemented in
cancelLoad()	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
extractDataSet(com.borland.dx.dataset.DataSet)	this class
extractDataSetChanges(com.borland.dx.dataset.DataSet)	this class
finalize()	java.lang.Object
hashCode()	java.lang.Object
loadDataSet(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

extractDataSet(com.borland.dx.dataset.DataSet)

public static DataSetData extractDataSet(DataSet dataSet)

Populates the *DataSetData* with data and non-transient data members that specify metadata information and status bits for each row. The metadata information includes the column count, row count, column names, data

types, *rowId*, and whether each *Column* is hidden. The status bits are used internally.

The data is organized in arrays of *Column* data. For example, if a data column is of type *Variant.INTEGER*, an *int* array is used for the values of that *Column*.

Any columns that don't already exist in the *DataSet* are added.

Note Physical types and properties such as *sqlType*, *tableName*, and *schemaName* are not contained in the *DataSetData*. These properties are not needed for editing purposes and should be extracted from the DBMS directly if needed.

extractDataSetChanges(*com.borland.dx.dataset.DataSet*)

```
public static DataSetData extractDataSetChanges(DataSet dataSet)
```

Similar to *extractDataSet* except that it extracts only the changes to the *DataSet* (edits, inserts, and deletes) that can then be sent to the server.

loadDataSet(*com.borland.dx.dataset.DataSet*)

```
public void loadDataSet(DataSet dataSet)
```

Loads the data contained in *DataSetData* and saves it to the destination *dataSet*, setting column properties such as *hidden* and *rowID*. Any columns that don't already exist in the destination *dataSet* area added.

DataSetException class

dx.dataset package

Extends *java.lang.Exception*

Extended by *com.borland.datastore.DataStoreException*,
com.borland.datastore.TxException,
com.borland.dx.dataset.ValidationException,
com.borland.dx.sql.dataset.ResolutionException

Implements *com.borland.jb.util.ChainedException*, *java.io.Serializable*

The *DataSetException* class extends the *java.lang.Exception* class and is used heavily by the *dataset* package. It encapsulates common errors that can be generated, and methods necessary to access this error information.

The *DataSetException* class can have other types of exceptions chained to them, for example, *java.io.IOException* and *java.sql.SQLException* exceptions. In these cases the *DataSetException* has an appropriate message that describes the error from the perspective of the DataExpress level API.

Programmatically, use the *getExceptionChain()* to obtain any chained exception. A chained exception (a singly linked list) can contain non-*DataSetException* exceptions that were encountered at a lower level API.

A *ValidationException* is generated by a constraint violation, for example, a minimum or maximum value outside specified ranges, a data entry that doesn't meet an edit mask specification, an attempt at updating a read-only column, and so on.

DataSetException variables

Variable	Defined in
ALREADY_LOADING	this class
BAD_PROCEDURE_PROPERTIES	this class
BAD_QUERY_PROPERTIES	this class
CANNOT_CHANGE_COLUMN	this class
CANNOT_CHANGE_COLUMN_DATA_TYPE	this class
CANNOT_FIND_TABLE_NAME	this class
CANNOT_IMPORT_NULL_DATASET	this class
CANNOT_REFRESH	this class
CANNOT_SAVE_CHANGES	this class
CANNOT_UPDATE_SCOPED_DATA_ROW	this class
CLASS_NOT_FOUND_ERROR	this class
COLUMN_ALREADY_BOUND	this class
COLUMN_NOT_IN_ROW	this class
COLUMN_TYPE_CONFLICT	this class
CONNECTION_DESCRIPTOR_NOT_SET	this class
CONNECTION_NOT_CLOSED	this class
DATA_FILE_LOAD_FAILED	this class
DATASET_CORRUPT	this class
DATASET_HAS_NO_ROWS	this class
DATASET_HAS_NO_TABLES	this class
DATASET_NOT_OPEN	this class
DATASET_OPEN	this class
DELETE_DUPLICATES	this class
DRIVER_NOT_LOADED_AT_RUNTIME	this class
DRIVER_NOT_LOADED_IN_DESIGN	this class
DUPLICATE_COLUMN_NAME	this class
DUPLICATE_PRIMARY	this class
EMPTY_COLUMN_NAMES	this class
errorCode	this class
EXCEPTION_CHAIN	this class
exceptionChain	this class
FIELD_POST_ERROR	this class
GENERIC_ERROR	this class
INCOMPATIBLE_DATA_ROW	this class

Variable	Defined in
INSUFFICIENT_ROWID	this class
INVALID_AGG_DESCRIPTOR	this class
INVALID_CLASS	this class
INVALID_COLUMN_POSITION	this class
INVALID_COLUMN_TYPE	this class
INVALID_DATA_FILE_FORMAT	this class
INVALID_FORMAT	this class
INVALID_ITERATOR_USE	this class
INVALID_SCHEMA_FILE	this class
INVALID_SORT_COLUMN	this class
INVALID_STORE_CLASS	this class
INVALID_STORE_NAME	this class
IO_ERROR	this class
LINK_COLUMNS_ERROR	this class
LINKFIELD_IN_USERPARAMETERS	this class
LOADING_NOT_STARTED	this class
MASTER_DETAIL_VIEW_ERROR	this class
MASTER_NAVIGATION_ERROR	this class
MISMATCH_PARAM_RESULT	this class
MISMATCHED_PARAMETER_FORMAT	this class
MISSING_MASTER_DATASET	this class
MISSING_REPLACESTOREROW	this class
MISSING_RESOLVER	this class
MULTIPLE_ROWS_AFFECTED	this class
NEED_LOCATE_START_OPTION	this class
NEED_PROCEDUREPROVIDER	this class
NEED_QUERYPROVIDER	this class
NEED_STORED_DATASET	this class
NEEDS_RECALC	this class
NO_CALC_AGG_FIELDS	this class
NO_CALC_FIELDS	this class
NO_DATABASE_TO_RESOLVE	this class
NO_NON_BLOB_COLUMNS	this class
NO_PRIMARY_KEY	this class
NO_PRIOR_ORIGINAL_ROW	this class
NO_RESULT_SET	this class
NO_ROWS_AFFECTED	this class
NO_UPDATABLE_COLUMNS	this class
NO_WHERE_CLAUSE	this class
NON_EXISTENT_ROWID	this class
NOT_DATABASE_RESOLVER	this class

Variable	Defined in
NOT_SELECT_QUERY	this class
NOT_UPDATEABLE	this class
NULL_COLUMN_NAME	this class
ONEPASS_INPUT_STREAM	this class
PARAMETER_COUNT_MISMATCH	this class
PARTIAL_SEARCH_FOR_STRING	this class
PROCEDURE_FAILED	this class
PROCEDURE_IN_PROCESS	this class
PROVIDER_FAILED	this class
PROVIDER_OWNED	this class
QUERY_FAILED	this class
QUERY_IN_PROCESS	this class
READ_ONLY_STORE	this class
REFRESHROW_NOT_SUPPORTED	this class
REOPEN_FAILURE	this class
RESOLVE_FAILED	this class
RESOLVE_IN_PROGRESS	this class
RESTRUCTURE_IN_PROGRESS	this class
SET_CALCULATED_FAILURE	this class
SQL_ERROR	this class
TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED	this class
UNEXPECTED_END_OF_QUERY	this class
UNKNOWN_COLUMN_NAME	this class
UNKNOWN_DETAIL_NAME	this class
UNKNOWN_PARAM_NAME	this class
UNRECOGNIZED_DATA_TYPE	this class
URL_NOT_FOUND	this class
URL_NOT_FOUND_IN_DESIGN	this class
WRONG_DATABASE	this class

ALREADY_LOADING

```
public static final int ALREADY_LOADING = BASE+28
```

The *DataSet* cannot be loaded or is not in load mode. Unexpected Internal error.

BAD_PROCEDURE_PROPERTIES

```
public static final int BAD_PROCEDURE_PROPERTIES = BASE+89
```

The *database* or *procedure* property is not set.

BAD_QUERY_PROPERTIES

```
public static final int BAD_QUERY_PROPERTIES = BASE+1
```

One or more *connection* properties of the *Database*, or the *query* properties of the *QueryDataSet* are not set.

CANNOT_CHANGE_COLUMN

```
public static final int CANNOT_CHANGE_COLUMN = BASE+70
```

Cannot make the requested change to the *Column*. Use *Column.clone()* before applying changes.

CANNOT_CHANGE_COLUMN_DATA_TYPE

```
public static final int CANNOT_CHANGE_COLUMN_DATA_TYPE = BASE+14
```

Cannot change the data type of the *Column* because the *Column* already contains data.

CANNOT_FIND_TABLE_NAME

```
public static final int CANNOT_FIND_TABLE_NAME = BASE+7
```

Cannot determine the table name for the *query* property of the *QueryDataSet* from metadata or from parsing the *query* property.

CANNOT_IMPORT_NULL_DATASET

```
public static final int CANNOT_IMPORT_NULL_DATASET = BASE+9
```

The *DataSet* must contain *Columns* before data can be imported.

CANNOT_REFRESH

```
public static final int CANNOT_REFRESH = BASE+77
```

Cannot refresh the current *DataSet*.

CANNOT_SAVE_CHANGES

```
public static final int CANNOT_SAVE_CHANGES = BASE+76
```

Cannot save changes in the current *DataSet*.

CANNOT_UPDATE_SCOPED_DATA_ROW

```
public static final int CANNOT_UPDATE_SCOPED_DATA_ROW = BASE+30
```

Cannot modify *Column* elements in a scoped *DataRow*. Create a new scoped *DataRow* if needed.

CLASS_NOT_FOUND_ERROR

public static final int CLASS_NOT_FOUND_ERROR = BASE+99

Class could not be found during serialization.

COLUMN_ALREADY_BOUND

public static final int COLUMN_ALREADY_BOUND = BASE+13

The *Column* is already bound to a *DataSet*. Use *Column.clone()* to add this column.

COLUMN_NOT_IN_ROW

public static final int COLUMN_NOT_IN_ROW = BASE+19

The *Column* is in the *DataSet* but was excluded from the row.

COLUMN_TYPE_CONFLICT

public static final int COLUMN_TYPE_CONFLICT = BASE+11

Failed to add a *Column* with the same name, but different data type as an existing *Column*.

CONNECTION_DESCRIPTOR_NOT_SET

public static final int CONNECTION_DESCRIPTOR_NOT_SET = BASE+45

Cannot open a *Database* unless the *ConnectionDescriptor* is set.

CONNECTION_NOT_CLOSED

public static final int CONNECTION_NOT_CLOSED = BASE+80

The connection could not be closed.

DATA_FILE_LOAD_FAILED

public static final int DATA_FILE_LOAD_FAILED = BASE+60

The *DataFile* load operation failed.

DATASET_CORRUPT

public static final int DATASET_CORRUPT = BASE+35

Operation failed. The *DataSet* structure is corrupt.

DATASET_HAS_NO_ROWS

```
public static final int DATASET_HAS_NO_ROWS = BASE+41
```

Invalid access on an empty *DataSet*. If working with a master-detail relationship where the master *DataSet* has no rows and the details are being fetched as needed, add persistent *Columns* to the detail *DataSet*. This allows the initialization logic to determine the *Columns* of the detail *DataSet*.

DATASET_HAS_NO_TABLES

```
public static final int DATASET_HAS_NO_TABLES = BASE+18
```

Save operation failed. None of the *DataSet* class *Columns* that are updateable have a table name.

DATASET_NOT_OPEN

```
public static final int DATASET_NOT_OPEN = BASE+42
```

Operation failed. The *DataSet* is not open.

DATASET_OPEN

```
public static final int DATASET_OPEN = BASE+39
```

Operation cannot be performed on an open *DataSet*. Close the *DataSet* first.

DELETE_DUPLICATES

```
public static final int DELETE_DUPLICATES = BASE+105
```

Delete existing duplicates before creating a new unique sort. Use the *StorageDataSet.deleteDuplicates()* method to delete the duplicates.

DRIVER_NOT_LOADED_AT_RUNTIME

```
public static final int DRIVER_NOT_LOADED_AT_RUNTIME = BASE+83
```

The specified driver could not be loaded. This could be a problem with the driver itself, or that the driver is not found on the classpath. Modify the classpath in the project properties.

DRIVER_NOT_LOADED_IN_DESIGN

```
public static final int DRIVER_NOT_LOADED_IN_DESIGN = BASE+82
```

The specified driver could not be loaded. This could be a problem with the driver itself, or that the driver is not found in the file: “\JBuilder\bin\JBuilder.ini” (looking for the path in the Djava.class.path line in the JavaVM_properties section). Exit JBuilder before editing this file.

DUPLICATE_COLUMN_NAME

public static final int DUPLICATE_COLUMN_NAME = BASE+36

Duplicate value for the *columnName* property in this *DataSet*.

DUPLICATE_PRIMARY

public static final int DUPLICATE_PRIMARY = BASE+110

Attempt to specify different primary key when one already exists.

EMPTY_COLUMN_NAMES

public static final int EMPTY_COLUMN_NAMES = BASE+20

List of *Columns* is **null** or empty.

errorCode

protected int errorCode

Stores the error code that describes the reason for this *DataSetException*.

EXCEPTION_CHAIN

public static final int EXCEPTION_CHAIN = BASE+47

Chained exception. To access the lower level exception(s), call *getExceptionChain()*. Call *printStackTrace* to display all exceptions in the chain.

exceptionChain

protected ExceptionChain exceptionChain

The *Exception* chain that can be traversed. This variable is returned by the *exceptionChain* property.

FIELD_POST_ERROR

public static final int FIELD_POST_ERROR = BASE+81

The field value cannot be posted. This error occurs when a *DataSet* requests that a data aware control post field values being edited and the field value cannot be posted (most likely due to a *ValidationException* error). There is no additional information available on why a post failed.

GENERIC_ERROR

public static final int GENERIC_ERROR = BASE+0

Undefined error.

INCOMPATIBLE_DATA_ROW

```
public static final int INCOMPATIBLE_DATA_ROW = BASE+31
```

Attempted to use a *DataRow* with a *DataSet* that are no longer synchronized. A *DataRow* must be created with the *DataSet* it is used with. If the structure of a *DataSet* changes, a new *DataRow* must be created for it.

INSUFFICIENT_ROWID

```
public static final int INSUFFICIENT_ROWID = BASE+73
```

The *DataSet.refetchRow()* method was called with an insufficient *rowId* (row identifier).

INVALID_AGG_DESCRIPTOR

```
public static final int INVALID_AGG_DESCRIPTOR = BASE+46
```

Operation failed. The *AggDescriptor* has no *groupColumns*, or some *groupColumns* do not exist in the *DataSet*.

INVALID_CLASS

```
public static final int INVALID_CLASS = BASE+106
```

The class is not valid.

INVALID_COLUMN_POSITION

```
public static final int INVALID_COLUMN_POSITION = BASE+12
```

The *Column* position is out of range.

INVALID_COLUMN_TYPE

```
public static final int INVALID_COLUMN_TYPE = BASE+49
```

Operation failed. Invalid *Column* data type for a *DataSet*.

INVALID_DATA_FILE_FORMAT

```
public static final int INVALID_DATA_FILE_FORMAT = BASE+32
```

Attempt load or save a *DataSet* with an invalid *dataFileFormat* property setting. See *DataFileFormat* for valid file format settings.

INVALID_FORMAT

```
public static final int INVALID_FORMAT = BASE+71
```

Column formatting error for default, min, or max values using edit mask or display format.

INVALID_ITERATOR_USE

public static final int INVALID_ITERATOR_USE = BASE+109

The iterator operation is not allowed on this row.

This exception is typically caused because:

- A write operation was attempted while iterating a *ReadRow*.
- A navigation, *insertRow*, or *post()* operation was attempted when not iterating a *DataSet*.

INVALID_SCHEMA_FILE

public static final int INVALID_SCHEMA_FILE = BASE+50

Invalid schema file.

INVALID_SORT_COLUMN

public static final int INVALID_SORT_COLUMN = BASE+48

Operation failed. Attempt to sort on non sortable *Column*, for example a *Binary Column*.

INVALID_STORE_CLASS

public static final int INVALID_STORE_CLASS = BASE+108

The operation failed. The table cannot be opened with the specified class; it must be opened with a different class.

INVALID_STORE_NAME

public static final int INVALID_STORE_NAME = BASE+104

The value specified for the *storeName* property must be one or more characters in length.

IO_ERROR

public static final int IO_ERROR = BASE+79

Operation failed due to an IO error.

LINK_COLUMNS_ERROR

public static final int LINK_COLUMNS_ERROR = BASE+33

The number and data type of *Columns* in the *MasterLinkColumns* and *DetailLinkColumns* must match.

LINKFIELD_IN_USERPARAMETERS

```
public static final int LINKFIELD_IN_USERPARAMETERS = BASE+86
```

The specified column in the *DetailLinkColumns* should not be included in the *parameterRow*.

LOADING_NOT_STARTED

```
public static final int LOADING_NOT_STARTED = BASE+29
```

Cannot load data into the *DataSet*. The *DataSet* is not in load mode.
Unexpected Internal error.

MASTER_DETAIL_VIEW_ERROR

```
public static final int MASTER_DETAIL_VIEW_ERROR = BASE+34
```

The *SortDescriptor keys* property setting is not compatible with the *DataSet linkColumns* property. The *keys* property must start with all the *Columns* specified in the *masterLinkColumns* and *detailColumns* properties.

MASTER_NAVIGATION_ERROR

```
public static final int MASTER_NAVIGATION_ERROR = BASE+43
```

Error in navigating master *DataSet*.

MISMATCH_PARAM_RESULT

```
public static final int MISMATCH_PARAM_RESULT = BASE+93
```

The output parameters of this procedure did not match the specification.

MISMATCHED_PARAMETER_FORMAT

```
public static final int MISMATCHED_PARAMETER_FORMAT = BASE+5
```

Cannot mix named parameters and '?' parameter markers.

MISSING_MASTER_DATASET

```
public static final int MISSING_MASTER_DATASET = BASE+65
```

Detail *DataSet* being resolved without a master *DataSet*.

MISSING_REPLACESTOREROW

```
public static final int MISSING_REPLACESTOREROW = BASE+102
```

Unexpected internal error.

MISSING_RESOLVER

public static final int MISSING_RESOLVER = BASE+64

The *Resolver* object is missing.

MULTIPLE_ROWS_AFFECTED

public static final int MULTIPLE_ROWS_AFFECTED = BASE+38

More than one row was affected by resolution query (either DELETE or UPDATE query).

NEED_LOCATE_START_OPTION

public static final int NEED_LOCATE_START_OPTION = BASE+24

Must specify one of *FIRST*, *LAST*, *NEXT*, *PRIOR* for locate operations. See *com.borland.dx.dataset.Locate*.

NEED_PROCEDUREPROVIDER

public static final int NEED_PROCEDUREPROVIDER = BASE+96

A *ProcedureProvider* is required.

NEED_QUERYPROVIDER

public static final int NEED_QUERYPROVIDER = BASE+95

A *QueryProvider* is required.

NEED_STORAGEDATASET

public static final int NEED_STORAGEDATASET = BASE+90

The *fetchAsNeeded* property cannot be set on the *masterLinkDescriptor* on a *DataSetView*.

NEEDS_RECALC

public static final int NEEDS_RECALC = BASE+107

The data set must be recalculated.

NO_CALC_AGG_FIELDS

public static final int NO_CALC_AGG_FIELDS = BASE+92

Operation failed. There is a *CalcAggFieldsListener*, but no *Columns* with a *calcType* property set to *AGGREGATE*.

NO_CALC_FIELDS

public static final int NO_CALC_FIELDS = BASE+44

Operation failed. There is a *CalcFieldsListener*, but no *Columns* with the *calcType* property set to *CALC*.

NO_DATABASE_TO_RESOLVE

public static final int NO_DATABASE_TO_RESOLVE = BASE+97

Cannot resolve data, since the *Database* property is not set on the *Resolver*.

NO_NON_BLOB_COLUMNS

public static final int NO_NON_BLOB_COLUMNS = BASE+21

Operation cannot be completed. The *DataSet* has no non-blob *Columns*.

NO_PRIMARY_KEY

public static final int NO_PRIMARY_KEY = BASE+111

Operation failed. There was an attempt to specify a primary sort without specifying any *sortKey* columns.

NO_PRIOR_ORIGINAL_ROW

public static final int NO_PRIOR_ORIGINAL_ROW = BASE+100

When an updated row is loaded by *StorageDataSet.loadRow(int status)*, the original row must be loaded immediately prior to the updated row.

NO_RESULT_SET

public static final int NO_RESULT_SET = BASE+78

Execution of query did not return a result set.

NO_ROWS_AFFECTED

public static final int NO_ROWS_AFFECTED = BASE+51

No row was affected by resolution query (either DELETE or UPDATE query).

NO_UPDATABLE_COLUMNS

public static final int NO_UPDATABLE_COLUMNS = BASE+8

Could not find any updateable columns when saving *DataSet* data.

NO_WHERE_CLAUSE

public static final int NO_WHERE_CLAUSE = BASE+40

A *QueryDataSet* that uses delayed detail fetching must have a where clause in its *query* property.

NON_EXISTENT_ROWID

public static final int NON_EXISTENT_ROWID = BASE+74

DataSet.refetchRow() was called with a non existent rowId (row identifier).

NOT_DATABASE_RESOLVER

public static final int NOT_DATABASE_RESOLVER = BASE+67

Trying to use the *Database* component to save changes to a *DataSet* with a non *DatabaseResolver* derived resolver.

NOT_SELECT_QUERY

public static final int NOT_SELECT_QUERY = BASE+2

Not a SELECT query, can't be parsed.

NOT_UPDATEABLE

public static final int NOT_UPDATEABLE = BASE+37

The *DataSet* has no unique row identifiers and is not updateable.

NULL_COLUMN_NAME

public static final int NULL_COLUMN_NAME = BASE+68

Trying to perform an operation that requires a column name without having specified one.

ONEPASS_INPUT_STREAM

public static final int ONEPASS_INPUT_STREAM = BASE+72

Attempting to read twice from a input stream that doesn't support *reset()*.

PARAMETER_COUNT_MISMATCH

public static final int PARAMETER_COUNT_MISMATCH = BASE+6

Mismatch between number of parameters markers in query and number of parameters in the *ReadRow*.

PARTIAL_SEARCH_FOR_STRING

public static final int PARTIAL_SEARCH_FOR_STRING = BASE+23

Partial search option can only be used when last column searched on is of *String* type.

PROCEDURE_FAILED

public static final int PROCEDURE_FAILED = BASE+103

Execution of the stored procedure failed.

PROCEDURE_IN_PROCESS

public static final int PROCEDURE_IN_PROCESS = BASE+94

A procedure is already in progress for this *DataSet*.

PROVIDER_FAILED

public static final int PROVIDER_FAILED = BASE+87

Execution of the *Provider* failed.

PROVIDER_OWNED

public static final int PROVIDER_OWNED = BASE+88

The *Provider* is already owned by another *DataSet*.

QUERY_FAILED

public static final int QUERY_FAILED = BASE+62

Execution of query failed.

QUERY_IN_PROCESS

public static final int QUERY_IN_PROCESS = BASE+26

A query is already in progress for this *DataSet*.

READ_ONLY_STORE

public static final int READ_ONLY_STORE = BASE+101

Store property is set to read only.

REFRESHROW_NOT_SUPPORTED

public static final int REFRESHROW_NOT_SUPPORTED = BASE+75

The current *DataSet* does not support a refetch row operation.

REOPEN_FAILURE

public static final int REOPEN_FAILURE = BASE+61

Failure notification to dependent components of a *DataSet* reopen.

RESOLVE_FAILED

public static final int RESOLVE_FAILED = BASE+63

Resolve failed.

RESOLVE_IN_PROGRESS

public static final int RESOLVE_IN_PROGRESS = BASE+69

The *DataSet* changes are currently being saved. Retry operation later.

RESTRUCTURE_IN_PROGRESS

public static final int RESTRUCTURE_IN_PROGRESS = BASE+27

Operation failed. A restructure operation is already in progress.

SET_CALCULATED_FAILURE

public static final int SET_CALCULATED_FAILURE = BASE+15

Cannot set the *calcType* property for *Columns* that already have data.

SQL_ERROR

public static final int SQL_ERROR = BASE+66

A *SQLException* from JDBC API.

TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED

public static final int TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED = BASE+22

Driver does not support this (or any higher) transaction isolation level.

UNEXPECTED_END_OF_QUERY

public static final int UNEXPECTED_END_OF_QUERY = BASE+3

Unexpected end of query; can't be parsed.

UNKNOWN_COLUMN_NAME

public static final int UNKNOWN_COLUMN_NAME = BASE+10

Cannot find a *Column* with the given *columnName*.

UNKNOWN_DETAIL_NAME

public static final int UNKNOWN_DETAIL_NAME = BASE+98

Cannot find the specified detail *DataSet*.

UNKNOWN_PARAM_NAME

public static final int UNKNOWN_PARAM_NAME = BASE+4

No matching named query parameter could be found.

UNRECOGNIZED_DATA_TYPE

public static final int UNRECOGNIZED_DATA_TYPE = BASE+16

Unknown *Variant* data type.

URL_NOT_FOUND

public static final int URL_NOT_FOUND = BASE+84

The specified URL could not be found. Check for misspellings, and that the correct driver is present on the classpath.

URL_NOT_FOUND_IN_DESIGN

public static final int URL_NOT_FOUND_IN_DESIGN = BASE+85

The specified URL could not be found. Check for misspellings, and that the correct driver is present on the class path.

WRONG_DATABASE

public static final int WRONG_DATABASE = BASE+91

The *database* property of the *ProcedureDescriptor* doesn't match the database being resolved to.

DataSetException constructors

DataSetException(int, java.lang.String)

public DataSetException(int errorCode, String message)

Creates a *DataSetException* object with a single error.

<i>errorCode</i>	The error code generated.
<i>message</i>	The text message associated with the error code.

DataSetException(int, java.lang.String, com.borland.jb.util.ExceptionChain)

public DataSetException(int errorCode, String message, ExceptionChain chain)

Creates a *DataSetException* object with properties as specified in its parameters.

<i>errorCode</i>	The error code generated.
<i>message</i>	The text message associated with the error code.
<i>chain</i>	The <i>ExceptionChain</i> object that contains the linked list of generated <i>Exception</i> objects.

DataSetException(int, java.lang.String, java.lang.Throwable)

public DataSetException(int errorCode, String message, Throwable ex)

Creates a *DataSetException* object with properties as specified in its parameters.

<i>message</i>	The text message associated with the error code.
<i>ex</i>	The <i>Exception</i> that was generated.

DataSetException(java.lang.String)

public DataSetException(String message)

Creates a *DataSetException* object with the specified message.

<i>message</i>	The text message associated with the error code.
----------------	--

DataSetException properties

Property	Implemented in
class*	java.lang.Object
errorCode*	this class
exceptionChain*	this class
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

errorCode

public int getErrorCode()

Read-only property that returns the error code associated with the *DataSetException*.

exceptionChain

public ExceptionChain getExceptionChain()

Read-only property that returns a chained *Exception* through a *com.borland.jb.util.ExceptionChain* object. The chained exception (a singly linked list) includes non-*DataSetExceptions* that were encountered at a lower level API.

DataSetException methods

Method	Implemented in
addExceptionListener (com.borland.dx.dataset.ExceptionListener)	this class
badProcedureProperties()	this class
badQueryProperties()	this class
classNotFoundException(java.lang.ClassNotFoundException)	this class
clone()	java.lang.Object
connectionDescriptorNotSet()	this class
connectionNotClosed(java.lang.Exception)	this class
dataSetHasNoTable()	this class
dataSetNotOpen()	this class
deleteDuplicates()	this class
driverNotLoadedAtRuntime(java.lang.String)	this class
driverNotLoadedInDesign(java.lang.String)	this class
equals(java.lang.Object)	java.lang.Object
fillInStackTrace()	java.lang.Throwable

Method	Implemented in
finalize()	java.lang.Object
getExceptionListeners()	this class
hashCode()	java.lang.Object
insufficientRowId()	this class
invalidClass(java.lang.Class)	this class
invalidClass(java.lang.String, java.lang.String)	this class
invalidColumnType(com.borland.dx.dataset.Column)	this class
invalidSQLType(int)	this class
invalidStoreName(java.lang.String)	this class
IOException(java.io.IOException)	this class
mismatchedParameterFormat()	this class
mismatchParamResult()	this class
missingMasterDataSet()	this class
mkUrlNotFound(java.lang.String, java.lang.Exception)	this class
mkUrlNotFoundInDesign(java.lang.String, java.lang.Exception)	this class
multipleRowsAffected(java.lang.String)	this class
needProcedureProvider()	this class
needQueryProvider()	this class
needsRecalc(java.lang.String)	this class
noDatabaseOnResolver()	this class
nonExistentRowId()	this class
noResultSet()	this class
noRowsAffected(java.lang.String)	this class
notDatabaseResolver()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
notSelectQuery()	this class
notSortable()	this class
noUpdatableColumns()	this class
noWhereClause(com.borland.dx.dataset.DataSet)	this class
onePassInputStream(com.borland.dx.dataset.Column)	this class
parameterCountMismatch(int, int, int)	this class
printStackTrace()	this class
printStackTrace(java.io.PrintStream)	this class
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
procedureFailed(java.lang.Exception)	this class
procedureInProgress()	this class
providerFailed(java.lang.Exception)	this class
providerOwned()	this class
queryFailed(java.lang.Exception)	this class

Method	Implemented in
queryInProgress()	this class
readOnlyStore(java.lang.String)	this class
removeExceptionListener (com.borland.dx.dataset.ExceptionListener)	this class
resolveFailed(java.lang.Exception)	this class
SQLException(java.sql.SQLException)	this class
throwException(int, java.lang.Exception)	this class
throwExceptionChain(java.lang.Throwable)	this class
toString()	java.lang.Throwable
transactionIsolationLevelNotSupported()	this class
unexpectedEndOfQuery()	this class
unknownColumnName(java.lang.String)	this class
unknownDetailName(java.lang.String)	this class
unknownParamName(java.lang.String)	this class
unrecognizedDataType()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
wrongDatabase()	this class

addExceptionListener(com.borland.dx.dataset.ExceptionListener)

public static final void addExceptionListener(ExceptionListener listener)

Adds a listener for *ExceptionListener* dispatches.

badProcedureProperties()

public static final void badProcedureProperties()

Creates and throws a *DataSetException* of *BAD_PROCEDURE_PROPERTIES* when one or more properties of the *ProcedureDescriptor* are invalid.

badQueryProperties()

public static final void badQueryProperties()

Creates and throws a *DataSetException* of *BAD_QUERY_PROPERTIES* when one or more properties of the *QueryDescriptor* are invalid.

classNotFoundException(java.lang.ClassNotFoundException)

public static final void classNotFoundException(ClassNotFoundException ex)

Creates and throws a *DataSetException* of *CLASS_NOT_FOUND_ERROR*.

connectionDescriptorNotSet()

public static final void connectionDescriptorNotSet()

Creates and throws a *DataSetException* of *CONNECTION_DESCRIPTOR_NOT_SET* when the *connection* properties associated with connecting to a *Database* are not set. These properties are stored in the *ConnectionDescriptor* object.

connectionNotClosed(java.lang.Exception)

public static final void connectionNotClosed(Exception ex)

Creates and throws a *DataSetException* of *CONNECTION_NOT_CLOSED*.

dataSetHasNoTable()

public static final void dataSetHasNoTable()

Creates and throws a *DataSetException* of *DATASET_HAS_NO_TABLES*.

dataSetNotOpen()

public static final void dataSetNotOpen()

Creates and throws a *DataSetException* of *DATASET_NOT_OPEN*.

deleteDuplicates()

public static final void deleteDuplicates()

Creates and throws a *DataSetException* of *DELETE_DUPLICATES*.

driverNotLoadedAtRuntime(java.lang.String)

public static final void driverNotLoadedAtRuntime(String driver)

Creates and throws a *DataSetException* of *DRIVER_NOT_LOADED_At_RUNTIME*.

driverNotLoadedInDesign(java.lang.String)

public static final void driverNotLoadedInDesign(String driver)

Creates and throws a *DataSetException* of *DRIVER_NOT_LOADED_IN_DESIGN*.

getExceptionListeners()

public static final EventMulticaster getExceptionListeners()

Return listeners added by *addExceptionListeners*.

insufficientRowId()

```
public static final void insufficientRowId()
```

Creates and throws a *DataSetException* of *INSUFFICIENT_ROWID*.

invalidClass(java.lang.Class)

```
public static final void invalidClass(Class javaClass)
```

Creates and throws a *DataSetException* of *INVALID_CLASS*.

invalidClass(java.lang.String, java.lang.String)

```
public static final void invalidClass(String storeClassName, String className)
```

Creates and throws a *DataSetException* of *INVALID_CLASS*.

invalidColumnType(com.borland.dx.dataset.Column)

```
public static final void invalidColumnType(Column column)
```

Creates and throws a *DataSetException* of *INVALID_COLUMN_TYPE*.

invalidSQLType(int)

```
public static final void invalidSQLType(int sqlType)
```

Creates and throws a *DataSetException* of *INVALID_COLUMN_TYPE*.

invalidStoreName(java.lang.String)

```
public static final void invalidStoreName(String name)
```

Creates and throws a *DataSetException* of *INVALID_STORE_NAME* when the value of the *storeName* property is blank.

IOException(java.io.IOException)

```
public static final void IOException(IOException ex)
```

Creates and throws an *IOError* exception.

mismatchedParameterFormat()

```
public static final void mismatchedParameterFormat()
```

Creates and throws a *DataSetException* of *MISMATCHED_PARAMETER_FORMAT*.

mismatchParamResult()

```
public static final void mismatchParamResult()
```

Creates and throws a *DataSetException* of *MISMATCH_PARAM_RESULT*.

missingMasterDataSet()

```
public static final void missingMasterDataSet()
```

Creates and throws a *DataSetException* of *MISSING_MASTER_DATASET*.

mkUrlNotFound(java.lang.String, java.lang.Exception)

```
public static final DataSetException mkUrlNotFound(String url, Exception ex)
```

Creates and throws a *DataSetException* of *URL_NOT_FOUND*.

mkUrlNotFoundInDesign(java.lang.String, java.lang.Exception)

```
public static final DataSetException mkUrlNotFoundInDesign(String url, Exception ex)
```

Creates and throws a *DataSetException* of *URL_NOT_FOUND_IN_DESIGN*.

multipleRowsAffected(java.lang.String)

```
public static final void multipleRowsAffected(String message)
```

Creates and throws a *DataSetException* of *MULTIPLE_ROWS_AFFECTED*.

needProcedureProvider()

```
public static final void needProcedureProvider()
```

Creates and throws a *DataSetException* of *NEED_PROCEDUREPROVIDER*.

needQueryProvider()

```
public static final void needQueryProvider()
```

Creates and throws a *DataSetException* of *NEED_QUERYPROVIDER*.

needsRecalc(java.lang.String)

```
public static final void needsRecalc(String storeName)
```

Creates and throws a *DataSetException* of *NEEDS_RECALC*.

noDatabaseOnResolver()

```
public static final void noDatabaseOnResolver()
```

Creates and throws a *DataSetException* of *NO_DATABASE_TO_RESOLVE*.

nonExistentRowId()

```
public static final void nonExistentRowId()
```

Creates and throws a *DataSetException* of *NON_EXISTENT_ROWID*.

noResultSet()

```
public static final void noResultSet()
```

Creates and throws a *DataSetException* of *NO_RESULT_SET*.

noRowsAffected(java.lang.String)

```
public static final void noRowsAffected(String message)
```

Creates and throws a *DataSetException* of *NO_ROWS_AFFECTED*.

notDatabaseResolver()

```
public static final void notDatabaseResolver()
```

Creates and throws a *DataSetException* of *NOT_DATABASE_RESOLVER*.

notSelectQuery()

```
public static final void notSelectQuery()
```

Creates and throws a *DataSetException* of *NOT_SELECT_QUERY*.

notSortable()

```
public static final void notSortable()
```

Creates and throws a *DataSetException* of *INVALID_SORT_COLUMN*.

noUpdatableColumns()

```
public static final void noUpdatableColumns()
```

Creates and throws a *DataSetException* of *NO_UPDATABLE_COLUMNS*.

noWhereClause(com.borland.dx.dataset.DataSet)

```
public static final void noWhereClause(DataSet dataSet)
```

Creates and throws a *DataSetException* of *NO_WHERE_CLAUSE*.

onePassInputStream(com.borland.dx.dataset.Column)

```
public static final void onePassInputStream(Column column)
```

Creates and throws a *DataSetException* of *ONEPASS_INPUT_STREAM*.

parameterCountMismatch(int, int, int)

public static final void parameterCountMismatch(int paramCount, int paramRowCount, int paramMasterCount)

Creates and throws a *DataSetException* of *PARAMETER_COUNT_MISMATCH*.

printStackTrace()

public void printStackTrace()

Prints the stack trace and the *ExceptionChain* object, if not **null**.

Overrides java.lang.Throwable.printStackTrace()

printStackTrace(java.io.PrintStream)

public void printStackTrace(java.io.PrintStream out)

Overrides the base class implementation of *printStackTrace()* to display the error code and any exceptions that may be chained to it.

Overrides java.lang.Throwable.printStackTrace(java.io.PrintStream)

procedureFailed(java.lang.Exception)

public static final void procedureFailed(Exception ex)

Creates and throws a *DataSetException* of *PROCEDURE_FAILED*.

procedureInProcess()

public static final void procedureInProcess()

Creates and throws a *DataSetException* of *PROCEDURE_IN_PROCESS*.

providerFailed(java.lang.Exception)

public static final void providerFailed(Exception ex)

Creates and throws a *DataSetException* of *PROVIDER_FAILED*.

providerOwned()

public static final DataSetException providerOwned()

Creates and throws a *DataSetException* of *PROVIDER_OWNED*.

queryFailed(java.lang.Exception)

public static final void queryFailed(Exception ex)

Creates and throws a *DataSetException* of *QUERY_FAILED*.

queryInProgress()

```
public static final void queryInProgress()
```

Creates and throws *DataSetException* of *QUERY_IN_PROCESS*.

readOnlyStore(java.lang.String)

```
public static final void readOnlyStore(String dataSetName)
```

Creates and throws a *DataSetException* of *READ_ONLY_STORE*.

removeExceptionListener(com.borland.dx.dataset.ExceptionListener)

```
public static final void removeExceptionListener(ExceptionListener listener)
```

Removes a listener for *ExceptionListener* dispatches.

resolveFailed(java.lang.Exception)

```
public static final void resolveFailed(Exception ex)
```

Creates and throws a *DataSetException* of *RESOLVE_FAILED*.

SQLException(java.sql.SQLException)

```
public static final void SQLException(SQLException ex)
```

Creates and throws a *DataSetException* of *SQL_ERROR*.

throwException(int, java.lang.Exception)

```
public static final void throwException(int errorCode, Exception ex)
```

Creates and throws a *DataSetException* of type *Exception.errorCode*. Extracts the corresponding String message for that error using *Exception.getMessage()*.

throwExceptionChain(java.lang.Throwable)

```
public static final void throwExceptionChain(Throwable ex)
```

Creates and throws a *DataSetException* of type *Throwable*.

transactionIsolationLevelNotSupported()

```
public static final void transactionIsolationLevelNotSupported()
```

Creates and throws *DataSetException* of *TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED*.

unexpectedEndOfQuery()

```
public static final void unexpectedEndOfQuery()
```

Creates and throws a *DataSetException* of *UNEXPECTED_END_OF_QUERY*.

unknownColumnName(java.lang.String)

```
public static final void unknownColumnName(String columnName)
```

Creates and throws a *DataSetException* of *UNKNOWN_COLUMN_NAME*.

unknownDetailName(java.lang.String)

```
public static final void unknownDetailName(String detailName)
```

Creates and throws a *DataSetException* of *UNKNOWN_DETAIL_NAME*.

unknownParamName(java.lang.String)

```
public static final void unknownParamName(String columnName)
```

Creates and throws a *DataSetException* of *UNKNOWN_PARAM_NAME*.

unrecognizedDataType()

```
public static final void unrecognizedDataType()
```

Creates and throws a *DataSetException* of *UNRECOGNIZED_DATA_TYPE*.

wrongDatabase()

```
public static final void wrongDatabase()
```

Creates and throws *DataSetException* of *WRONG_DATABASE*.

DataSetView component

dx.dataset package

Extends com.borland.dx.dataset.DataSet

Implements com.borland.dx.dataset.AccessListener, com.borland.dx.dataset.Designable, com.borland.dx.dataset.MasterNavigateListener, com.borland.dx.dataset.StatusListener, java.io.Serializable, java.util.EventListener

The *DataSetView* component extends *DataSet* functionality by presenting an alternate view of the data in the *DataSet*. The *DataSetView* itself has no storage of data but sees all the unfiltered data contained in its *storageDataSet* property to which you can apply a different sort order and filter criterion

than the original *StorageDataSet*. The navigation of the *DataSetView* data is separate from that of its *StorageDataSet*.

The use of a *DataSetView* component is optional. If you only need a single sort or filter criterion at a time in your application, you may apply those settings directly on the *QueryDataSet*, *ProcedureDataSet*, or *TableDataSet*. If however you wish to present the data in the *DataSet* in multiple “views”, the *DataSetView* provides that capability without the need for multiple objects that each store data.

When a *DataSetView* shares the data storage of a *DataSet* with another *DataSetView*, each *DataSetView* sees edits that both make to the data. Calling the *saveChanges()* method saves changes that were made to both. Both also share column properties such as edit and display masks, and so on. You cannot, however, display the *DataSetView* component’s columns nor set its column properties (including persistent) programmatically. This is because the *DataSetView* component doesn’t have its own data storage.

The *DataSetView* component also allows for an additional level of indirection which provides for greater flexibility when changing the binding of your UI components. If you anticipate the need to rebind your UI components and have several of them, bind the components to a *DataSetView* instead of directly to the *StorageDataSet*. When you need to rebind, change the *DataSetView* component to the appropriate *StorageDataSet*, thereby making a single change that affects all UI components connected to the *DataSetView* as well.

Warning

The *DataSetView* component extends the functionality provided by its superclass (*DataSet*). The *close()* method is inherited by *DataSetView* and has particular importance with this component as it must be called to ensure that the component is garbage collected. Otherwise, a *DataSetView* component cannot be garbage collected until its associated *StorageDataSet* is garbage collected.

DataSetView constructors

DataSetView()

public DataSetView()

Creates a *DataSetView* component.

DataSetView properties

Property	Implemented in
assignedNull**	com.borland.dx.dataset.ReadWriteRow
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow

Property	Implemented in
columns*	com.borland.dx.dataset.ReadRow
defaultValues**	com.borland.dx.dataset.DataSet
detailDataSetWithFetchAsNeeded*	com.borland.dx.dataset.DataSet
details*	com.borland.dx.dataset.DataSet
displayErrors	com.borland.dx.dataset.DataSet
editable	com.borland.dx.dataset.DataSet
editing*	com.borland.dx.dataset.DataSet
editingNewRow*	com.borland.dx.dataset.DataSet
empty*	com.borland.dx.dataset.DataSet
enableDelete	com.borland.dx.dataset.DataSet
enableInsert	com.borland.dx.dataset.DataSet
enableUpdate	com.borland.dx.dataset.DataSet
internalRow*	com.borland.dx.dataset.DataSet
lastColumnVisited	com.borland.dx.dataset.DataSet
masterLink	this class
open*	com.borland.dx.dataset.DataSet
row*	com.borland.dx.dataset.DataSet
rowCount*	com.borland.dx.dataset.DataSet
rowFilterListener*	com.borland.dx.dataset.DataSet
schemaName*	com.borland.dx.dataset.DataSet
sort	com.borland.dx.dataset.DataSet
status*	com.borland.dx.dataset.DataSet
storageDataSet**	this class
tableName*	com.borland.dx.dataset.DataSet
unassignedNull**	com.borland.dx.dataset.ReadWriteRow

masterLink

```
public final MasterLinkDescriptor getMasterLink()
public synchronized void setMasterLink(MasterLinkDescriptor descriptor)
```

Specifies the *MasterLinkDescriptor* object that describes the relationship between this *DataSetView* and another *DataSet*. The *get* method is defined in *DataSet*.

storageDataSet

```
public synchronized void setStorageDataSet(StorageDataSet dataSetStore)
```

Specifies the *StorageDataSet* object that contains the data storage that the *DataSetView* component accesses. Use the *storageDataSet* property of the *DataSet* class to set this property. The *get* method is defined in *DataSet*.

DataSetView methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	com.borland.dx.dataset.DataSet
addRow(com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
allocateValues()	com.borland.dx.dataset.DataSet
atFirst()	com.borland.dx.dataset.DataSet
atLast()	com.borland.dx.dataset.DataSet
cancel()	com.borland.dx.dataset.DataSet
cancelLoading()	com.borland.dx.dataset.DataSet
cancelOperation()	com.borland.dx.dataset.DataSet
canNavigate (com.borland.dx.dataset.Column, int)	com.borland.dx.dataset.DataSet
canSet(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
clearStatus()	com.borland.dx.dataset.DataSet
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneDataSetView()	com.borland.dx.dataset.DataSet
close()	com.borland.dx.dataset.DataSet
columnIsVisible(java.lang.String)	com.borland.dx.dataset.DataSet
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	com.borland.dx.dataset.DataSet
deleteRow()	com.borland.dx.dataset.DataSet
dittoRow(boolean, boolean)	com.borland.dx.dataset.DataSet
dittoRow(boolean)	com.borland.dx.dataset.DataSet
dropIndex()	com.borland.dx.dataset.DataSet
editRow()	com.borland.dx.dataset.DataSet
emptyAllRows()	com.borland.dx.dataset.DataSet
emptyRow()	com.borland.dx.dataset.DataSet
enableDataSetEvents(boolean)	com.borland.dx.dataset.DataSet
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow

Method	Implemented in
first()	com.borland.dx.dataset.DataSet
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDataRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDataRow(int, com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDetail(java.lang.String)	com.borland.dx.dataset.DataSet
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
goToClosestRow(int)	com.borland.dx.dataset.DataSet
goToInternalRow(long)	com.borland.dx.dataset.DataSet
goToRow(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
goToRow(int)	com.borland.dx.dataset.DataSet
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	com.borland.dx.dataset.DataSet
hashCode()	java.lang.Object
hasValidations()	com.borland.dx.dataset.DataSet
inBounds()	com.borland.dx.dataset.DataSet
insertRow(boolean)	com.borland.dx.dataset.DataSet
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	com.borland.dx.dataset.DataSet
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.DataSet
isModified(java.lang.String)	com.borland.dx.dataset.DataSet
isNew(int)	com.borland.dx.dataset.DataSet
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	com.borland.dx.dataset.DataSet
locate(com.borland.dx.dataset.ReadRow, int)	com.borland.dx.dataset.DataSet
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	com.borland.dx.dataset.DataSet
masterNavigated(com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
masterNavigating(com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
moveRow(int)	com.borland.dx.dataset.DataSet

Method	Implemented in
next()	com.borland.dx.dataset.DataSet
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.dx.dataset.DataSet
openDetails()	com.borland.dx.dataset.DataSet
post()	com.borland.dx.dataset.DataSet
postAllDataSets()	com.borland.dx.dataset.DataSet
prior()	com.borland.dx.dataset.DataSet
refetchRow (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.DataSet
refilter()	com.borland.dx.dataset.DataSet
refresh()	com.borland.dx.dataset.DataSet
refreshSupported()	com.borland.dx.dataset.DataSet
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
resetInBounds()	com.borland.dx.dataset.DataSet
resetPendingStatus(boolean)	com.borland.dx.dataset.DataSet
resetPendingStatus(long, boolean)	com.borland.dx.dataset.DataSet
saveChanges()	com.borland.dx.dataset.DataSet
saveChangesSupported()	com.borland.dx.dataset.DataSet
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.DataSet
setDisplayVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startEditCheck (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
statusMessage (com.borland.dx.dataset.StatusEvent)	com.borland.dx.dataset.DataSet
statusMessage(int, java.lang.String)	com.borland.dx.dataset.DataSet
toggleViewOrder(java.lang.String)	com.borland.dx.dataset.DataSet
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
validate()	com.borland.dx.dataset.DataSet
validate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

DataSetView event listeners

This component is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

DxDispatch interface

dx.dataset package

Implemented by com.borland.dx.dataset.MasterNavigateEvent
An interface for dispatchable events.

DxDispatch methods

Method	Implemented in
dxDispatch(java.util.EventListener)	this class

dxDispatch(java.util.EventListener)

void dxDispatch(EventListener listener)

An abstract dispatch method.

listener The listener the event is sent to.

EditAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.EditListener, java.util.EventListener

This is an adapter class for *EditListener*, which is used as a notification for row editing before and after edit-related operations for *DataSets* are completed.

EditAdapter properties

Property	Implemented in
class*	java.lang.Object

EditAdapter methods

Method	Implemented in
added(com.borland.dx.dataset.DataSet)	this class
addError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)	this class
adding(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	this class
canceling(com.borland.dx.dataset.DataSet)	this class
clone()	java.lang.Object
deleted(com.borland.dx.dataset.DataSet)	this class

Method	Implemented in
<code>deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>deleting(com.borland.dx.dataset.DataSet)</code>	this class
<code>editError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>equals(java.lang.Object)</code>	java.lang.Object
<code>finalize()</code>	java.lang.Object
<code>hashCode()</code>	java.lang.Object
<code>inserted(com.borland.dx.dataset.DataSet)</code>	this class
<code>inserting(com.borland.dx.dataset.DataSet)</code>	this class
<code>modifying(com.borland.dx.dataset.DataSet)</code>	this class
<code>notify()</code>	java.lang.Object
<code>notifyAll()</code>	java.lang.Object
<code>toString()</code>	java.lang.Object
<code>updated(com.borland.dx.dataset.DataSet)</code>	this class
<code>updateError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>updating(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow)</code>	this class
<code>wait()</code>	java.lang.Object
<code>wait(long, int)</code>	java.lang.Object
<code>wait(long)</code>	java.lang.Object

EditListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.EditAdapter

This interface is used as a notification for row editing before and after edit-related operations are completed and includes

- Methods that occur before a row is posted (methods ending in “ing”)
- Methods that occur after a row is posted (methods ending in “ed”)
- Methods that occur when an exception is thrown in response to an edit (methods ending in “Error”)

With an *EditListener*, you can

- Block adding, deleting, or updating of rows. This is useful when different users have different access rights, or have rights that are dependent on data values. Note that the *DataSet* *enable*, *insert*, *delete*, and *update* properties can be used to block certain types of editing.
- Perform row-level validation just before a row is posted. Use the *adding()* method for new rows and the *updating()* method for modified rows.
- Get control after one of these operations. For example, you can initialize values in fields after a row is inserted, but before the user begins data entry.
- Process a *ValidationException* for all events. An exception class that derives from *Exception* can be thrown. These exceptions are caught by the *DataSet* event dispatcher and chained into a special *ValidationException*. This *ValidationException* has an error code of *ValidationException.APPLICATION_ERROR* and the message from the exception that was caught. This chained *ValidationException* is then thrown so that normal *DataSet* error handling can deal with the problem.

Inserting, adding, and updating methods each have a unique purpose. Insert methods create a new, unposted row. The new, unposted row is sometimes called a *pseudo-row* because it does not exist in the data set until it is posted. Add methods work on newly inserted rows when they are about to be or have been posted. Update methods work on existing rows only, at the time that modifications to them are about to be or have been posted.

A simple way for an application to pass an error message to display in the UI involves the *EditListener* before event (those that end in “ing”). These events can be wired to throw a *Exception* (“custom message”). In turn, this gets thrown as a chained *ValidationException* that copies the “custom message” as the message for the *ValidationException*. Since all *ValidationExceptions* go to a *StatusListener*, the custom message displays in the application’s UI.

EditListener methods

Method	Implemented in
<code>added(com.borland.dx.dataset.DataSet)</code>	this class
<code>addError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>adding(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)</code>	this class
<code>canceling(com.borland.dx.dataset.DataSet)</code>	this class
<code>deleted(com.borland.dx.dataset.DataSet)</code>	this class

Method	Implemented in
<code>deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>deleting(com.borland.dx.dataset.DataSet)</code>	this class
<code>editError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>inserted(com.borland.dx.dataset.DataSet)</code>	this class
<code>inserting(com.borland.dx.dataset.DataSet)</code>	this class
<code>modifying(com.borland.dx.dataset.DataSet)</code>	this class
<code>updated(com.borland.dx.dataset.DataSet)</code>	this class
<code>updateError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>updating(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow)</code>	this class

added(com.borland.dx.dataset.DataSet)

`void added(DataSet dataSet)`

This is an event to notify listeners that a new row is successfully posted to the *DataSet*. This event is fired by *DataSet.addRow()*, which inserts, modifies, and posts a row all in one operation.

dataSet The data set the row was added to.

addError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)

`void addError(DataSet dataSet, ReadWriteRow row, DataSetException ex, ErrorResponse response)`

This is an event to notify listeners when an exception is thrown for row add operations. Call *response.abort()* (the default) to cause the operation to fail with an appropriate *DataSetException* or *ValidationException*. Call *response.retry()* to cause the operation to be retried. Be sure that the retry will succeed or that your code can handle repeated retries. Call *response.ignore()* to cause the operation to silently fail without an exception being thrown.

dataSet The data set that has the error.

row The row that contains the error.

<i>ex</i>	The type of exception that was thrown.
<i>response</i>	The type of response to the error.

adding(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.ReadWriteRow`)

`void adding(DataSet dataSet, ReadWriteRow newRow)`

This is an event to notify listeners before a new row is posted to the *DataSet*. This event is fired by *DataSet.addRow()*, which inserts, modifies, and posts a row all in one operation. If a *VetoException* or *Exception* is thrown inside this method, the post operation is not performed, a *ValidationException* with an error code of `APPLICATION_ERROR` is thrown instead. The *adding()* method is called before checks to make sure all required fields are not null. If a *VetoException* or *Exception* is constructed with a `STRING` parameter, this `STRING` is used in the default error handling displays, for example,

```
throw new VetoException("My error message");
```

<i>dataSet</i>	The <i>DataSet</i> that the row will be posted to.
<i>newRow</i>	The row that is to be inserted, modified, and posted.

canceled(`com.borland.dx.dataset.DataSet`)

`void canceled(DataSet dataSet)`

This is an event to notify listeners when the editing of a new or existing row in a *DataSet* is about to be canceled. An application might use this event to save undo information.

<i>dataSet</i>	The data set to which edits are about to be canceled.
----------------	---

deleted(`com.borland.dx.dataset.DataSet`)

`void deleted(DataSet dataSet)`

This is an event to notify listeners that a successful delete operation has been performed.

<i>dataSet</i>	The data set from which rows have been deleted.
----------------	---

deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)

```
void deleteError(DataSet dataSet, DataSetException ex, ErrorResponse response)
```

This is an event to notify listeners when an exception is thrown for row delete operations. Call *response.abort()* (the default) to causes the operation to fail with an appropriate *DataSetException* or *ValidationException*. Call *response.retry()* to cause the operation to be retried. Be sure that the retry will succeed or that your code can handle repeated retries. Call *response.ignore()* to cause the operation to silently fail without throwing an exception.

<i>dataSet</i>	The data set that has the error.
<i>ex</i>	The type of exception that was thrown.
<i>response</i>	The type of response to the error.

deleting(com.borland.dx.dataset.DataSet)

```
void deleting(DataSet dataSet)
```

This is an event to notify listeners before a row is deleted from the *DataSet*. If a *VetoException* or *Exception* is thrown inside this method, the delete operation is not performed, a *ValidationException* with an error code of *APPLICATION_ERROR* is thrown instead. If a *VetoException* or *Exception* is constructed with a *STRING* parameter, this *STRING* is used in the default error handling displays, for example,

```
throw new VetoException("My error message");
```

<i>dataSet</i>	The data set from which a row is about to be deleted.
----------------	---

editError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Variant, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)

```
void editError(DataSet dataSet, Column column, Variant value, DataSetException ex, ErrorResponse response)
```

This is an event to notify listeners when any exceptions occur setting a column value. This includes validation check failures as well as any *VetoExceptions* thrown by a *ColumnChangeListener.validating()* event handler. The *ErrorResponse* object allows the user to indicate how the error should be handled. Call *response.abort()* (the default) to cause the operation to fail with an appropriate *DataSetException* or *ValidationException*. Call *response.retry()* to cause the operation to be retried. Be sure that the retry will succeed or that

your code can handle repeated retries. Call *response.ignore()* to cause the operation to silently fail without an exception being thrown.

<i>dataSet</i>	The data set that has the error.
<i>column</i>	The column that contains the error.
<i>value</i>	The value that causes the error.
<i>ex</i>	The type of exception that was thrown.
<i>response</i>	The type of response to the error.

inserted(com.borland.dx.dataset.DataSet)

void inserted(DataSet dataSet)

This is an event to notify listeners that a new, unposted row is inserted into the *DataSet*. This event can be used to initialize row values of new rows.

<i>dataSet</i>	The data set to which the row has just been inserted.
----------------	---

inserting(com.borland.dx.dataset.DataSet)

void inserting(DataSet dataSet)

This is an event to notify listeners just before a *DataSet* attempts to insert a new, unposted row. If a *VetoException* or *Exception* is thrown inside this method, the insert operation is not performed, a *ValidationException* with an error code of APPLICATION_ERROR is thrown instead. If a *VetoException* or *Exception* is constructed with a STRING parameter, this STRING is used in the default error handling displays, for example,

```
throw new VetoException("My error message");
```

<i>dataSet</i>	The data set to which the row is about to be inserted.
----------------	--

modifying(com.borland.dx.dataset.DataSet)

void modifying(DataSet dataSet)

This is an event to notify listeners when a user begins to modify an existing row. If a *VetoException* or *Exception* is thrown inside this method, the modify operation is not performed, a *ValidationException* with an error code of APPLICATION_ERROR is thrown instead. If a *VetoException* or *Exception* is constructed with a STRING parameter, this STRING is used in the default error handling displays, for example,

```
throw new VetoException("My error message");
```

<i>dataSet</i>	The data set that contains the row being modified.
----------------	--

updated(com.borland.dx.dataset.DataSet)

void updated(DataSet dataSet)

This is an event to notify listeners that a modified row has been successfully posted to a *DataSet*.

dataSet The data set to which the modified row has been posted.

**updateError(com.borland.dx.dataset.DataSet,
com.borland.dx.dataset.ReadWriteRow,
com.borland.dx.dataset.DataSetException,
com.borland.jb.util.ErrorResponse)**

void updateError(DataSet dataSet, ReadWriteRow row, DataSetException ex, ErrorResponse response)

This is an event to notify listeners when an exception is thrown for row changes. Call *response.abort()* (the default) to cause the operation to fail with an appropriate *DataSetException* or *ValidationException*. Call *response.retry()* to cause the operation to be retried. Be sure that the retry will succeed or that your code can handle repeated retries. Call *response.ignore()* to cause the operation to silently fail without an exception being thrown.

dataSet The data set that has the error.

row The row that contains the error.

ex The type of exception that was thrown.

response The type of response to the error.

**updating(com.borland.dx.dataset.DataSet,
com.borland.dx.dataset.ReadWriteRow,
com.borland.dx.dataset.ReadRow)**

void updating(DataSet dataSet, ReadWriteRow newRow, ReadRow oldRow)

This is an event to notify listeners before a modified row is posted to the *DataSet*. If an exception is thrown inside this method, the post operation is not performed, a *ValidationException* with an error code of APPLICATION_ERROR is thrown instead. The *updating()* method is called before checks to make sure all required fields are not null. If a *VetoException* or *Exception* is constructed with a STRING parameter, this STRING is used in the default error handling displays, for example,

```
throw new VetoException("My error message");
```

dataSet The data set to which a modified row is about to be posted.

newRow The row containing the modified data.

oldRow The row containing the data in the row prior to modification.

ExceptionEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

This class is used to override the *DataSet* error handling for data-aware controls.

ExceptionEvent variables

Variable	Defined in
source	java.util.EventObject

ExceptionEvent constructors

ExceptionEvent(com.borland.dx.dataset.DataSet, java.awt.Component, java.lang.Throwable)

public ExceptionEvent(DataSet dataSet, Component component, Throwable ex)

Constructs an *ExceptionEvent* object with the following parameters:

<i>dataSet</i>	The <i>DataSet</i> being accessed when this error occurred.
<i>component</i>	The <i>Component</i> on which the exception occurred.
<i>ex</i>	The <i>Exception</i> that occurred.

ExceptionEvent properties

Property	Implemented in
class*	java.lang.Object
component*	this class
dataSet*	this class
exception*	this class
exceptionChain*	com.borland.jb.util.DispatchableEvent
source*	java.util.EventObject

component

public Component getComponent()

Component that was being accessed when the exception occurred. Can be **null**.

dataSet

public DataSet getDataSet()

DataSet that was being accessed when the exception occurred. Can be **null**.

exception

public Throwable getException()

Exception that occurred.

ExceptionEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

This method is an implementation of *DispatchableEvent* that an *EventMulticaster* uses to dispatch an event of this type to the listener.

listener The listener to dispatch this event to.

See also *com.borland.jb.util.DispatchableEvent*, *com.borland.jb.util.EventMulticaster*

Overrides *com.borland.jb.util.DispatchableEvent.dispatch(java.util.EventListener)*

ExceptionListener interface

dx.dataset package

Extends java.util.EventListener

This interface is implemented for notification that an exception is generated. This is generally implemented by applications that want centralized control over the handling of *DataSetExceptions* encountered by data-aware controls. When a data-aware control gets an exception from a data set, the default *handleException* method has control and may display a dialog box with the exception or may treat it as a *ValidationException*. This class is used to allow you to decide how to handle the exception.

The reference to this listener is statically stored in the *DataSet* class. If the *DataSet* class is ever garbage collected, the registration of this listener will be lost. To prevent this, you can hold a reference to the *DataSet* class with the following statement:

```
Void main(String args){
    Object classHolder = DataSet.class;
}
```

ExceptionListener methods

Method	Implemented in
exception(com.borland.dx.dataset.ExceptionEvent)	this class

exception(com.borland.dx.dataset.ExceptionEvent)

public void exception(ExceptionEvent event)

Called when a *DataSetException* occurs.

event An *ExceptionEvent* object that tells what exception occurred and the data set and component where it occurred.

LoadCancel interface

dx.dataset package

Implemented by com.borland.dx.dataset.DataSetData, com.borland.dx.dataset.TextDataFile, com.borland.dx.sql.dataset.JdbcProvider, com.borland.dx.sql.dataset.OracleProcedureProvider, com.borland.dx.sql.dataset.ProcedureProvider, com.borland.dx.sql.dataset.QueryProvider

This interface is generally called internally by a *DataSet* component's *cancelLoading()* method, however, you might implement this interface if you are writing your own loader using the *startLoading(com.borland.dx.dataset.LoadCancel, int, boolean)*, *loadRow()*, and *endLoading()* method of a *StorageDataSet*.

A *QueryDataSet* has a *cancelLoading()* method. When you call the *cancelLoading()* method, the data set asks if it supports the *LoadCancel* interface. If so, the *cancelLoading()* method then calls the *LoadCancel* interface's *cancelLoad()* method to cancel a load operation. This class is especially useful for asynchronous queries and can also be used to cancel a *TextDataFile* load operation.

LoadCancel methods

Method	Implemented in
<i>cancelLoad()</i>	this class

cancelLoad()

public void cancelLoad()

Cancels a load operation on a *DataSet*.

LoadEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

This class is used as a notification that a load operation on a *StorageDataSet* has been completed. Load operations occur when a query or procedure is executed, or when a *StorageDataSet* is loaded from an import operation. This notification is most interesting for queries and procedures that are executed with asynchronous fetching, since this is done with a separate thread.

LoadEvent variables

Variable	Defined in
source	java.util.EventObject

LoadEvent constructors

LoadEvent(java.lang.Object)

public LoadEvent(Object source)

Constructs a *LoadEvent* object.

LoadEvent properties

Property	Implemented in
class*	java.lang.Object
exceptionChain*	com.borland.jb.util.DispatchableEvent
source*	java.util.EventObject

LoadEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

Sends a notification of the *LoadEvent* to all registered listeners.

Overrides com.borland.jb.util.DispatchableEvent.dispatch(EventListener)

LoadListener interface

dx.dataset package

Extends java.util.EventListener

This interface is used as a notification that a load operation on a *StorageDataSet* has been completed. Currently load operations occur when a query or procedure is executed, and when a *StorageDataSet* is loaded from an import operation. This notification is most interesting for queries and procedures that are executed with asynchronous fetching, since this is done with a separate thread.

This is particularly useful with operations on more than one thread. Use this interface when you want notification that all data has been fetched before you run an operation, such as a row count, against the data set.

LoadListener methods

Method	Implemented in
dataLoaded(com.borland.dx.dataset.LoadEvent)	this class

dataLoaded(com.borland.dx.dataset.LoadEvent)

public void dataLoaded(LoadEvent event)

This is an event to notify the listener that a *StorageDataSet* has been loaded.

event The event that has occurred, in this case a *LoadEvent*, indicating that a *StorageDataSet* has been loaded.

LoadRowListener interface

dx.dataset package

Extends java.util.EventListener

If a class implementing this interface is wired to a *StorageDataSet* by calling the *StorageDataSet.addLoadRowListener()* method, then the *loadRow* implementation will be called for every row that is loaded into the *StorageDataSet*.

Rows are “loaded” into a *StorageDataSet* by calling *StorageDataSet.loadRow()*.

StorageDataSet.Provider implementations and *DataSetData* use *StorageDataSet.loadRow()* to quick populate a *StorageDataSet*.

LoadRowListener methods

Method	Implemented in
loadRow(int, com.borland.dx.dataset.ReadWriteRow)	this class

loadRow(int, com.borland.dx.dataset.ReadWriteRow)

public void loadRow(int status, ReadWriteRow row)

This method is called for each row that is loaded into a *StorageDataSet*.

- status

Status of the row. Pre-defined constants for this property are listed under *RowStatus* variables.
- row

The row about to be loaded into the *StorageDataSet*.

Locate interface

dx.dataset package

The *Locate* interface encapsulates the most commonly-used options when performing a search operation. It allows you to specify how a particular row is found when using the *DataSet.locate()* method and the *DataSet.lookup()* method. For example, options include case sensitivity, search for the first, subsequent or last occurrence, and so on.

The *Locate* variables may be combined where it makes sense to do so. For example, you can search using partial strings and specifying case insensitivity. Combine variables using the Java bitwise OR operator of a vertical pipe symbol (|) between each variable.

Locate variables

Variable	Defined in
CASE_INSENSITIVE	this class
DETAIL	this class
FAST	this class
FIRST	this class
LAST	this class
NEXT	this class
NEXT_FAST	this class
PARTIAL	this class
PRIOR	this class

Variable	Defined in
PRIOR_FAST	this class
START_MASK	this class

CASE_INSENSITIVE

public static final int CASE_INSENSITIVE = 0x8

Search ignoring upper or lower case differences. Valid only for *String* columns.

DETAIL

static final int DETAIL = 0x100

This variable is used internally by other *com.borland* classes. You should never use this variable directly.

FAST

public static final int FAST = 0x80

Use search values from the previous search (instead of initializing new values). This option offers quicker performance since new values are not initialized.

FIRST

public static final int FIRST = 0x20

Locate the first occurrence.

LAST

public static final int LAST = 0x40

Locate the last occurrence

NEXT

public static final int NEXT = 0x2

Search from the current row position.

NEXT_FAST

public static final int NEXT_FAST = NEXT|FAST

Search from the current row position, using values from the previous search.

PARTIAL

```
public static final int PARTIAL = 0x1
```

Allow partial matches for *Columns* containing *String* values. The columns are specified in the locate method in the order that they are searched (and not in *Column* order of the *DataSet*). This option works only on the last column specified, and only for *String* columns.

PRIOR

```
public static final int PRIOR = 0x4
```

Search backwards from the current position.

PRIOR_FAST

```
public static final int PRIOR_FAST = PRIOR|FAST
```

Search backwards from the current position using values from the previous search.

START_MASK

```
static final int START_MASK = FIRST|LAST|NEXT|PRIOR)
```

This variable is used internally by other *com.borland* classes. You should never use this variable directly.

MasterLinkDescriptor class

dx.dataset package

Extends java.lang.Object

Implements java.io.Serializable

The *MasterLinkDescriptor* object stores properties that set a master-detail relationship between two *DataSet* objects such as *QueryDataSet*, *ProcedureDataSet*, *TableDataSet* or *DataSetView*. You can link different *DataSet* objects together, for example, a *QueryDataSet* and a *TableDataSet* as long as there is common data to base the link relationship on. *JDataStore* creates indexes for the linking relationship as needed, quickly and efficiently.

Mechanisms for fetching detail data

There are two methods of using a *StorageDataSet* to provide the detail rows in a master-detail relationship:

- **immediate fetching:** where a single query is used to fetch the entire detail set in one batch. After the initial execution of the query and fetching of the

results, the two *DataSets* are linked based on the cached data, using settings stored in the *MasterLinkDescriptor*.

- **delayed fetching:** is a less consistent view of the data. Each time a master row is visited for the first time, its detail data is fetched by re-executing the detail query. The detail query must contain a WHERE clause with links to the master *DataSet* using named parameters. The detail *DataSet*'s *loadRow()* method is invoked each time a master row is first encountered. This method extracts the link columns data values from the master (as specified in the *MasterLinkDescriptor*) and binds them to the query. Then the query executes and the results of that query added to any already cached rows in the detail *DataSet*. There must be a one-to-one match of the master *DataSet* columns names to the named parameters in the WHERE clause of the detail *QueryDataSet*.

When specifying the query statement for the detail *DataSet*, you should specify the linking columns in a *where* clause if *fetchAsNeeded* is **true**. If the *fetchAsNeeded* setting is not compatible with the detail query, you will get a *DataSetException* and anomalous results.

For example, to bind parameters in the correct order where

- Master table ("Master") contains fields: MasterColumn1, MasterColumn2, MasterColumn3
- Detail table ("Detail") contains fields: DetailColumn1, DetailColumn2, DetailColumn3
- Relationship is Master.MasterColumn1 to Detail.DetailColumn2

The correct query statement for the detail query is:

```
SELECT * FROM DETAIL WHERE DetailColumn2 = :MasterColumn1
```

Programmatically calling the *MasterLinkDescriptor*

To work with the *MasterLinkDescriptor* class programmatically, you set its properties when instantiating the *MasterLinkDescriptor* object. There are no write accessors for this class. The properties of the *MasterLinkDescriptor* object are:

- *fetchAsNeeded*
- *detailLinkColumns*
- *masterDataSet*
- *masterLinkColumns*
- *cascadeDeletes*
- *cascadeUpdates*

These properties are required on the detail *DataSet* only when setting up a master-detail relationship.

fetchAsNeeded for detail DataSets

When specifying the query statement for the detail *DataSet*, you should specify the linking columns in a *where* clause if *fetchAsNeeded* is **true**. If the *fetchAsNeeded* setting is not compatible with the detail query, you will get a *DataSetException* and anomalous results.

How editing operations affect multiple rows of a data set

This section discusses how editing operations that affect multiple rows of a data set work on detail data sets. Only *refresh()* varies according to how the *fetchAsNeeded* property is set.

- The *saveChanges()* method (and the Save button on a *JdbNavToolBar*) saves changes to all details sets for all masters, whether *fetchAsNeeded* is **true** or **false**.
- The *refresh()* method (and the Refresh button on a *JdbNavToolBar*) performs differently according to *fetchAsNeeded*. If *fetchAsNeeded* is **false**, it refreshes all detail sets; if **true**, it refreshes the details for the current master row only. This is consistent with the fact that detail sets are fetched at different times when *fetchAsNeeded* is **true**, or are never fetched if their masters are never visited, so they are also refreshed independently.
- The *empty()* method deletes all details for all master rows, whether *fetchAsNeeded* is **true** or **false**.
- The *emptyAllRows()* method deletes all details for the current master row only, whether *fetchAsNeeded* is **true** or **false**. This is because *emptyAllRows()* affects visible rows only, and only one detail set is visible at a time.

Self-joins

Self-joins (where a *DataSet* is linked to itself) are not supported in *JDataStore*.

Cascading updates and deletes

To cascade updates and deletes, set the *cascadeUpdates* and *cascadeDeletes* properties as appropriate. These properties should be used with care, especially in cases where you have multiple master-detail relationships chained where a detail *DataSet* is a master to another detail (and so on). Typically in those circumstances, you want the updates to be reflected through the chain of related *DataSets*. For this to be possible, you must set the *cascadeUpdates* and *cascadeDeletes* properties at each level. Otherwise, the cascading stops at the *DataSet* whose *cascade* property is **false**.

Partial updates or deletions (orphan records) can also occur in cases where a *DataSet* somewhere in the master-detail chain is not updateable. Program logic may also cause this property to effect only partial updates. For instance, an event handler for the *editListener*'s deleting event might allow deletion of

some detail rows and block deletion of others. In the case of cascaded updates, you may end up with orphan details if some rows in a detail set can be updated and others can't.

Displaying all detail data when a master-detail relationship is in effect

In a master-detail relationship, visible detail rows include only those which match the current master. To see all detail rows, instantiate a *DataSetView* component using the detail *DataSet* as the data source. Navigation through the data in the *DataSetView* is also independent of that of the detail *DataSet*. To turn the master-detail relationship off, you can also call *DataSet.setMasterLink(null)*.

Empty master *DataSet*

In establishing a master-detail relationship using *fetchAsNeeded*, *JDataStore* uses the master *DataSet* columns in order to run the query that obtains the data for the detail *DataSet*. If there are no rows in the master *DataSet*, the detail query cannot be run.

An application can ensure that the detail query is always executable by setting up persistent columns for the master and detail data sets which include all respective linking columns.

This is especially important for a *QueryDataSet* that has a *MasterLinkDescriptor* with the *fetchAsNeeded* property set to true. In this case, *DataExpress* cannot determine what columns are present in the detail because its query cannot be run without parameter values from the current row of the master. Setting persistent columns for the detail linking columns will allow the detail data set to be opened.

MasterLinkDescriptor constructors

MasterLinkDescriptor(*com.borland.dx.dataset.DataSet*, *java.lang.String*[], *java.lang.String*[])

```
public MasterLinkDescriptor(DataSet masterDataSet, String[] masterLinkColumns, String[]
    detailLinkColumns)
```

Constructs a *MasterLinkDescriptor* with properties as specified in its parameters.

<i>masterDataSet</i>	The <i>DataSet</i> which is the master of the master-detail relationship.
<i>masterLinkColumns</i>	An array of <i>String</i> column names from the <i>masterDataSet</i> to use when linking.
<i>detailLinkColumns</i>	An array of <i>String</i> column names from the detail <i>DataSet</i> to use when linking.

MasterLinkDescriptor(**com.borland.dx.dataset.DataSet**, **java.lang.String[]**, **java.lang.String[]**, **boolean**)

```
public MasterLinkDescriptor(DataSet masterDataSet, String[] masterLinkColumns, String[]
    detailLinkColumns, boolean fetchAsNeeded)
```

Constructs a *MasterLinkDescriptor* with properties as specified in its parameters.

<i>masterDataSet</i>	The <i>DataSet</i> which is the master of the master-detail relationship.
<i>masterLinkColumns</i>	An array of <i>String</i> column names from the <i>masterDataSet</i> to use when linking.
<i>detailLinkColumns</i>	An array of <i>String</i> column names from the detail <i>DataSet</i> to use when linking.
<i>fetchAsNeeded</i>	A boolean value which determines whether detail data is fetched all at once or fetched when a master is navigated to a row whose details have not yet been fetched. For more information on this property, see <i>fetchAsNeeded</i> .

MasterLinkDescriptor(**com.borland.dx.dataset.DataSet**, **java.lang.String[]**, **java.lang.String[]**, **boolean**, **boolean**, **boolean**)

```
public MasterLinkDescriptor(DataSet masterDataSet, String[] masterLinkColumns, String[]
    detailLinkColumns, boolean fetchAsNeeded, boolean cascadeUpdates, boolean cascadeDeletes)
```

Constructs a *MasterLinkDescriptor* with properties as specified in its parameters.

<i>masterDataSet</i>	The <i>DataSet</i> which is the master of the master/detail relationship.
<i>masterLinkColumns</i>	An array of column names to use from master the master <i>DataSet</i> .
<i>detailLinkColumns</i>	An array of column names to use from the detail <i>DataSet</i> .
<i>fetchAsNeeded</i>	If true, causes details to be fetched as the master <i>DataSet</i> navigates. If the <i>QueryDataSet</i> or <i>ProcedureDataSet</i> components are being used, the detail query will be re-executed as the master is navigated.

<i>cascadeUpdates</i>	If true, updates to linking columns in the master <i>DataSet</i> are also cascaded to applicable rows in the detail <i>DataSet</i> . If false, updates to linking columns are not allowed if the master has details: they would be orphaned by this operation.
<i>cascadeDeletes</i>	If true, matching detail rows are deleted when the corresponding master <i>DataSet</i> row is deleted. If false, master rows that have details cannot be deleted.

MasterLinkDescriptor properties

Property	Implemented in
<i>cascadeDeletes*</i>	this class
<i>cascadeUpdates*</i>	this class
<i>class*</i>	java.lang.Object
<i>detailLinkColumns*</i>	this class
<i>fetchAsNeeded*</i>	this class
<i>masterDataSet*</i>	this class
<i>masterLinkColumns*</i>	this class

cascadeDeletes

public boolean isCascadeDeletes()

Read-only property that returns whether matching detail (*DataSet*) rows should be deleted when the corresponding row in the master *DataSet* is deleted. If **true**, deletes are also applied to matching details; if **false**, master rows that have details cannot be deleted. This property defaults to **false** meaning that deletes are not cascaded.

cascadeUpdates

public boolean isCascadeUpdates()

Read-only property that returns whether updates to linking columns in the master *DataSet* are also applied to rows in detail *DataSets*. If **true**, updates are cascaded to details; if **false**, updates to linking columns are not allowed if the master has details: the details would be orphaned by this operation. This property defaults to **false** meaning that updates are not cascaded.

detailLinkColumns

```
public String[] getDetailLinkColumns()
```

Read-only property that returns a *String* array containing the names of the column or columns of the detail *DataSet* that are used to link to the *MasterDataSet*. An index on the link columns of the detail *DataSet* is not required. The names of the link columns between the master and detail data sets do not need to match as long as the number and data types of linking columns do match.

Set this property when creating the *MasterLinkDescriptor* object by calling a *MasterLinkDescriptor* constructor that takes this property as a parameter.

fetchAsNeeded

```
public boolean isFetchAsNeeded()
```

Read-only property that determines whether the detail rows are all fetched at one time, or when a master is navigated to whose details have not yet been fetched.

Set this property when creating the *MasterLinkDescriptor* object by calling a *MasterLinkDescriptor* constructor that takes this property as a parameter.

Set this property to **true** when you want to fetch the detail data when needed. This is useful when you don't need to access the data for most details, when the detail data set is very large, or when you want to see the most current information for that detail set. The disadvantage is that the data may not be consistent since detail fetching is done in separate transactions.

When **true**, the detail rows for the corresponding master are fetched and stored in the detail *DataSet*. As additional details are fetched for other master rows, the corresponding detail data is added to the same detail *DataSet* but filtered so that you only see the details for the current master. If the detail data has already been fetched for a particular master and you want to get a refresh from the data source, you must post any changes in the detail data set first; otherwise, updates will be lost.

If this property is **true**, there should be a WHERE clause in the query string of the detail *DataSet*. If the WHERE clause is omitted, a *DataSetException* of *NO_WHERE_CLAUSE* is generated.

Set this property to **false** to retrieve all detail data at one time so that the detail data is a consistent snapshot at the time of the fetch. This is useful when working off-line (when not directly connected to the data source), or when data consistency across all details is an issue. For example, in a situation where an update affects multiple rows of data, you may want to fetch all details at one time instead of as-needed to ensure that the updated data is consistent across all affected rows. (This assumes some level of transaction processing when changes are saved back to the data source.)

This property is used for *QueryDataSet* and *ProcedureDataSet* components. It has no effect for *TableDataSet* components since all detail rows are fetched at one time.

masterDataSet

```
public DataSet getMasterDataSet()
```

Read-only property that returns the instantiated *DataSet* object that is the master in the relationship. The current position in the *MasterDataSet* determines what data is visible in the detail data set. As you navigate through the data in the *MasterDataSet*, the corresponding data (based on matches in the linking columns) is displayed for the detail *DataSet*.

Set this property when creating the *MasterLinkDescriptor* object by calling a *MasterLinkDescriptor* constructor that takes this property as a parameter.

masterLinkColumns

```
public String[] getMasterLinkColumns()
```

Read-only property that returns a *String* array of columns of the master *DataSet* that are used to link to a *DetailDataSet*. An index on the link columns of the master *DataSet* is not required. The names of the link columns between the master and detail data sets do not need to match as long as the data types of the linking columns do match.

Set this property when creating the *MasterLinkDescriptor* object by calling a *MasterLinkDescriptor* constructor that takes this property as a parameter.

MasterLinkDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

MaxAggOperator class

dx.dataset package

Extends com.borland.dx.dataset.BoundsAggOperator

Implements java.io.Serializable, java.lang.Cloneable

The *MaxAggOperator* component is an instantiatable subclass of the *AggOperator* and defines an aggregation operation where the maximum value in a *DataSet* is identified.

Set the *aggOperator* property of the *AggDescriptor* object to this class to identify the maximum value in a *DataSet*, using the property settings stored in the *AggDescriptor*. Attach the *AggDescriptor* object to a *Column* component's *agg* property to access the data that the aggregation uses.

MaxAggOperator variables

Variable	Defined in
aggColumn	com.borland.dx.dataset.AggOperator
aggDataSet	com.borland.dx.dataset.AggOperator
aggValue	com.borland.dx.dataset.AggOperator
dataSet	com.borland.dx.dataset.AggOperator
resultColumn	com.borland.dx.dataset.AggOperator
resultValue	com.borland.dx.dataset.AggOperator

MaxAggOperator properties

Property	Implemented in
class*	java.lang.Object

MaxAggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	com.borland.dx.dataset.BoundsAggOperator
clone()	com.borland.dx.dataset.AggOperator
delete (com.borland.dx.dataset.ReadRow, long)	com.borland.dx.dataset.BoundsAggOperator
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object

Method	Implemented in
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	com.borland.dx.dataset.BoundsAggOperator
locate (com.borland.dx.dataset.ReadRow)	this class
needsAggDataSet()	com.borland.dx.dataset.BoundsAggOperator
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.BoundsAggOperator
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

locate(com.borland.dx.dataset.ReadRow)

public boolean locate(ReadRow row)

Locates the maximum value for the grouping column values specified in *row*. For example, if the grouping column is region and the max column is sales, calling this method moves to the highest sales value for a region.

Overrides com.borland.dx.dataset.AggOperator.locate(com.borland.dx.dataset.ReadRow)

MetaDataUpdate interface

dx.dataset package

The *MetaDataUpdate* interface defines constants used with the *metaDataUpdate* property of the *QueryDataSet* and *ProcedureDataSet* components. Its constants specify whether metadata discovery should be performed (or not) when executing a query or stored procedure against a SQL server database.

If the driver used to connect to a SQL database does not support the *metaData* functions used by this package, you may get an error message indicating that edits to the data cannot be saved because none of its updateable columns have a table name. In such cases, try setting the *metaDataUpdate* property of the *QueryDataSet* or *ProcedureDataSet* to *MetaDataUpdate.NONE*. This bypasses the step of automatic querying for metadata information, however, in order for the data to be updateable, this information must be provided by the application. For *Column* components,

set the *rowID*, *precision*, *scale*, and *searchable* properties. For *QueryDataSet* and *ProcedureDataSet* components, set the *tableName* and *schemaName* properties.

MetaDataUpdate variables

Variable	Defined in
ALL	this class
NONE	this class
PRECISION	this class
ROWID	this class
SCALE	this class
SEARCHABLE	this class
TABLERNAME	this class

ALL

public static final int ALL = 31

This constant is the default and specifies that the open of the *DataSet* will automatically override the following settings:

- The *tableName* and *schemaName* properties of the *StorageDataSet*.
- The *rowId*, *precision*, *scale*, and *searchable* properties for the *Column* component.

NONE

public static final int NONE = 0

This constant is used to specify that the open of the *DataSet* is not to override the above stated properties. You must specify the values for these properties for any persistent *Column* components in the *StorageDataSet*.

PRECISION

public static final int PRECISION = 4

Specifies that the *precision* property of persistent columns should be overridden by the value detected in the driver's metadata.

ROWID

public static final int ROWID = 2

Specifies that a query should be analyzed for updateability. If set, the query string may be automatically changed to include columns that can be used to identify a row in a table. The *rowID* property is set or reset on all columns, overriding the settings in any persistent columns. The default resolver

(QueryResolver) needs this information to make update queries. This constant has no effect for *ProcedureDataSet* components.

SCALE

```
public static final int SCALE = 8
```

Specifies that the *scale* property of persistent columns should be overridden by the value detected in the driver's metadata.

SEARCHABLE

```
public static final int SEARCHABLE = 16
```

Specifies that the *searchable* property of persistent columns should be overridden by the value detected in the driver's metadata.

TABlename

```
public static final int TABlename = 1
```

Specifies that the *tableName* and *schemaName* properties of the *StorageDataSet* and any persistent columns should be set when opening the *DataSet*. The default resolver (*Queryresolver*), needs this information to make update queries. This constant has no effect for *ProcedureDataSet* components.

MinAggOperator class

dx.dataset package

Extends com.borland.dx.dataset.BoundsAggOperator

Implements java.io.Serializable, java.lang.Cloneable

The *MinAggOperator* class is an instantiatable subclass of the *AggOperator* and defines an aggregation operation where the minimum value is determined.

Set the *aggOperator* property of the *AggDescriptor* object to this class to identify the minimum value in a *DataSet*, using the property settings stored in the *AggDescriptor*. Attach the *AggDescriptor* object to a *Column* component's *agg* property to access the data that the aggregation uses.

MinAggOperator variables

Variable	Defined in
aggColumn	com.borland.dx.dataset.AggOperator
aggDataSet	com.borland.dx.dataset.AggOperator

Variable	Defined in
aggValue	com.borland.dx.dataset.AggOperator
dataSet	com.borland.dx.dataset.AggOperator
resultColumn	com.borland.dx.dataset.AggOperator
resultValue	com.borland.dx.dataset.AggOperator

MinAggOperator properties

Property	Implemented in
class*	java.lang.Object

MinAggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	com.borland.dx.dataset.BoundsAggOperator
clone()	com.borland.dx.dataset.AggOperator
delete (com.borland.dx.dataset.ReadRow, long)	com.borland.dx.dataset.BoundsAggOperator
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	com.borland.dx.dataset.BoundsAggOperator
locate (com.borland.dx.dataset.ReadRow)	this class
needsAggDataSet()	com.borland.dx.dataset.BoundsAggOperator
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.BoundsAggOperator
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

locate(com.borland.dx.dataset.ReadRow)

```
public boolean locate(ReadRow row)
```

Locates the minimum value for the grouping column values specified in *row*. For example, if the grouping column is region and the min column is sales, calling this method moves to the lowest sales value for a region.

Overrides `com.borland.dx.dataset.AggregOperator.locate(com.borland.dx.dataset.ReadRow)`

NavigationEvent class

dx.dataset package

Extends `com.borland.jb.util.DispatchableEvent`

Implements `java.io.Serializable`

This class is used to provide notification that the *DataSet*'s cursor position has changed. This class is used by data-aware controls so that the UI can respond to the change in cursor location.

This event is sent out any time the current row position changes due to navigation operations like *first*, *last*, *next*, *prior*, and for editing operations like *deleteRow* and *insertRow*. Also note that *post()* can cause navigation since *post()* will navigate the newly posted row to its correct position in the *DataSet*. For sorted *DataSets* the posted position is determined by sort order. For non-sorted *DataSets* the posted position is at the end of the *DataSet*.

NavigationEvent variables

Variable	Defined in
source	java.util.EventObject

NavigationEvent constructors

NavigationEvent(java.lang.Object)

```
public NavigationEvent(Object source)
```

Creates a *NavigationEvent* object.

NavigationEvent properties

Property	Implemented in
class*	java.lang.Object
exceptionChain*	com.borland.jb.util.DispatchableEvent
source*	java.util.EventObject

NavigationEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

This method is an implementation of *DispatchableEvent* that an *EventMulticaster* uses to dispatch an event of this type to the listener.

listener The listener to dispatch this event to.

See also *com.borland.jb.util.DispatchableEvent, com.borland.jb.util.EventMulticaster*

Overrides *com.borland.jb.util.DispatchableEvent.dispatch(EventListener)*

NavigationListener interface

dx.dataset package

Extends java.util.EventListener

This interface is used as a notification that current row position has changed, or *navigated*, from a row. This is used for data-aware controls so that the UI can respond to the change in cursor location. For example, a status bar can change its display from “1 of 10” to “2 of 10” when the user moves from the first row to the second.

NavigationListener methods

Method	Implemented in
navigated(com.borland.dx.dataset.NavigationEvent)	this class

navigated(com.borland.dx.dataset.NavigationEvent)

public void navigated(NavigationEvent event)

This is an event to notify listeners that the current row has changed, that a user has moved from one row to another.

event

The event that called the listener, indicating in this case that the current row has been changed.

OpenAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.OpenListener, java.util.EventListener

This is an adapter class for *OpenListener*, which provides notification that a *DataSet* is opening, has opened, is closing, or has closed.

OpenAdapter properties

Property	Implemented in
class*	java.lang.Object

OpenAdapter methods

Method	Implemented in
clone()	java.lang.Object
closed(com.borland.dx.dataset.DataSet)	this class
closing(com.borland.dx.dataset.DataSet)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
opened(com.borland.dx.dataset.DataSet)	this class
opening(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

OpenListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.OpenAdapter

This interface is used as a notification when a *DataSet* is opened or closed. This is especially useful when your application uses many lookup tables. You may verify whether or not a *DataSet* is open before performing a lookup, open it as necessary to perform the lookup, then close it to improve performance.

OpenListener methods

Method	Implemented in
closed(com.borland.dx.dataset.DataSet)	this class
closing(com.borland.dx.dataset.DataSet)	this class
opened(com.borland.dx.dataset.DataSet)	this class
opening(com.borland.dx.dataset.DataSet)	this class

closed(com.borland.dx.dataset.DataSet)

```
public void closed(DataSet dataSet)
```

Called to notify the listener that a data set has been successfully closed.

dataSet The name of the data set that was closed.

closing(com.borland.dx.dataset.DataSet)

```
public void closing(DataSet dataSet)
```

Called to notify the listener before a data set is closed.

dataSet The name of the data set to be closed.

opened(com.borland.dx.dataset.DataSet)

```
public void opened(DataSet dataSet)
```

Called to notify the listener that a data set has been opened.

dataSet The name of the data set that was opened.

opening(com.borland.dx.dataset.DataSet)

```
public void opening(DataSet dataSet)
```

Called to notify the listener before a data set is opened.

dataSet The name of the data set to be opened.

ParameterRow component

dx.dataset package

Extends com.borland.dx.dataset.ReadWriteRow

Implements com.borland.dx.dataset.ColumnDesigner,
com.borland.dx.dataset.Designable, java.io.Serializable

The *ParameterRow* component is useful when working with parameter values for SQL statements of *QueryDataSet* and *ProcedureDataSet* components. To use this component in your application

- 1 instantiate a *ParameterRow*
- 2 add *Column* components
- 3 set any *Column* properties you need
- 4 set the *parameterRow* property of the *QueryDataSet* or *ProcedureDataSet* to this *ParameterRow*

Parameters can also be specified using a “scoped” *DataRow* that contains only some of the *Columns* in the associated *DataSet*. However, the *Column* components in the *DataRow* map directly to the *Columns* in the *DataSet*, and therefore do not allow more than once reference in the *DataRow*. The *ParameterRow* component allows you to specify the same column multiple times, for example, for range comparisons.

For example, you may want to specify query parameters which involve two or more range comparisons against the same *Column*. For the following query statement:

```
SELECT * FROM employee WHERE emp_no>=:LOW AND emp_no<=:HIGH
```

your *ParameterRow* should have a *Column* for each of its parameter names, :LOW and :HIGH. Place values you want the query to use in these *Columns* and associate them to the query (set the *parameterRow* property of the *QueryDataSet* or *ProcedureDataSet* to this *ParameterRow*. In this way, whenever you execute your query, you can use different values for :LOW and :HIGH without having to write multiple queries for each permutation.

ParameterRow constructors

ParameterRow()

```
public ParameterRow()
```

Default constructor that creates a *ParameterRow* component.

ParameterRow properties

Property	Implemented in
assignedNull**	com.borland.dx.dataset.ReadWriteRow
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns	this class
unassignedNull**	com.borland.dx.dataset.ReadWriteRow

columns

```
public Column[] getColumnns()
public void setColumns(Column[] columns)
```

Stores the *Column* components as a *String* array of *Column* names.

ParameterRow methods

Method	Implemented in
addColumn(com.borland.dx.dataset.Column, int)	this class
addColumn(com.borland.dx.dataset.Column)	this class
addColumn(java.lang.String, int, int)	this class
addColumn(java.lang.String, int)	this class
changeColumn(int, com.borland.dx.dataset.Column)	this class
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
dropColumn(java.lang.String)	this class
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hashCode()	java.lang.Object
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.ReadRow
isModified(java.lang.String)	com.borland.dx.dataset.ReadRow
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
notify()	java.lang.Object
notifyAll()	java.lang.Object
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.ReadWriteRow
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
toString()	com.borland.dx.dataset.ReadRow

Method	Implemented in
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

addColumn(com.borland.dx.dataset.Column, int)

public int addColumn(Column column, int parameterType)

Adds the specified *Column* to the *ParameterRow*, sets its *parameterType* as specified, then returns the ordinal position of the newly added *Column*. This method is a shortcut for setting the *parameterType* property on a *Column* and then calling the *addColumn(Column)* method. On error this method throws a *DataSetException*.

- column*

parameterType
- The *Column* component to add to this *ParameterRow*.

The usage type of the parameter. Valid values are defined in *ParameterType* variables.

addColumn(java.lang.String, int)

public void addColumn(String columnName, int dataType)

Adds a *Column* to the *ParameterRow*, sets its *columnName* as specified, and its *dataType* to *ParameterType.IN*. This method is useful for parameterized queries and is a short-cut to calling other *addColumn(...)* methods then setting the *parameterType* to *ParameterType.IN*. On error, this method throws a *DataSetException*.

- columnName*

dataType
- The *Column* component (specified by its String name) to add to this *ParameterRow*.

The data type of the parameter. Valid values are data type constants defined in *com.borland.dx.dataset.Variant* variables.

addColumn(java.lang.String, int, int)

public void addColumn(String columnName, int dataType, int parameterType)

Adds a *Column* to the *ParameterRow*, then sets its *columnName*, *dataType*, and *parameterType* properties as specified by its parameters. On error, this method throws a *DataSetException*.

<i>columnName</i>	The <i>Column</i> component (specified by its String name) to add to this <i>ParameterRow</i> .
<i>dataType</i>	The data type of the parameter. Valid values are data type constants defined in <i>com.borland.dx.dataset.Variant</i> variables.
<i>parameterType</i>	The usage type of the parameter. Valid values are defined in <i>ParameterType</i> variables.

ParameterType class

dx.dataset package

Extends java.lang.Object

This class defines constants that are used by *Column* objects in a *ParameterRow*. These constants indicate how the parameters are to be handled by the database server when the associated *QueryDataSet* or *ProcedureDataSet* is executed:

- for input only (and therefore read-only)
- output only
- both input and output
- a result set
- a return value from a stored procedure
- unknown

These constants have corresponding values defined in *java.sql.DatabaseMetaData* as *procedureColumnIn*, *procedureColumnInOut*, and so on.

ParameterType variables

Variable	Defined in
IN	this class
IN_OUT	this class
NONE	this class
OUT	this class
RESULT	this class
RETURN	this class

IN

public static final int IN = 1

Indicates that this parameter is used only for input and will not be modified.

IN_OUT

public static final int IN_OUT = 2

Indicates that this parameter is used both for input and for output.

NONE

public static final int NONE = 0

Indicates that this parameter type is unknown.

OUT

public static final int OUT = 4

Indicates that this parameter is used only for output.

RESULT

public static final int RESULT = 3

Indicates that this parameter is a result set.

RETURN

public static final int RETURN = 5

Indicates that this parameter is the returned value from a procedure.

ParameterType properties

Property	Implemented in
class*	java.lang.Object

ParameterType methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object

Method	Implemented in
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

PickListDescriptor class

dx.dataset package

Extends java.lang.Object

Implements java.io.Serializable

The *PickListDescriptor* describes a pick list relationship between a *Column* of one *DataSet* (the *target DataSet*), and a second *DataSet* (the *source DataSet*) that provides values from which you can choose. The following properties are stored in the *PickListDescriptor*:

- *pickListDataSet* specifies the *DataSet* that contains the items to display in the pick list.
- *pickListDisplayColumns* specifies the *Column* components of the *DataSet* to display in the pick list.
- *pickListColumns* specifies the columns of the *pickListDataSet* from which values in the selected row are copied to *destinationColumns*.
- *destinationColumns* specifies the *Column* components of the target *DataSet* that are populated with the values associated with the selected pick list choice.
- *enforceIntegrity* determines whether data integrity rules are enforced on the data added to *destinationColumns*. This property is not currently used.
- *lookupDisplayColumn* specifies which column to display (field display values) when the source data is not open. This property is used when the displayed items list differs from the values stored when an item is selected.

There are three basic ways to configure the *pickList* property, the first two are the more common uses:

- 1 **A simple “fill in”.** For visual controls, the pick list provides a selection of values from a *pickListDataSet*. The selected values are filled into the specified columns.

For this configuration, set the *pickListDataSet*, *destinationColumns* (columns to copy to), *pickListColumns* (columns to copy from the *pickListDataSet*),

and *pickListDisplayColumns* (columns values to display from the *pickListDataSet* when the *pickList* column is being edited).

- 2 **“Fill in” and “lookup” display.** Provides the same functionality as above, except you can specify a *lookupDisplayColumn* from the *pickListDataSet*. This allows you to store a “code number” code in the column, but display a “code description”.

The values from the *lookupDisplayColumn* in the *pickListDataSet* are displayed instead of the values stored in the *pickListColumns*. A value to display is chosen by using the current values in *destinationColumns* to locate the row in the *pickListDataSet* that has the same values for its *pickListColumns*.

If a match is found, the *lookupDisplayColumn* value for that row in the *pickListDataSet* is displayed. Note that the *DataSet.setValue()* and *getValue()* methods retrieve and set the “real” value stored. To retrieve the display value for such a column, use the *DataSet.getDisplayVariant()* method.

- 3 **Dynamically created read-only “lookup”.** This configuration is a little more work to setup, but allows an application to show both the code and code description values dynamically. You can achieve a similar effect with the first configuration above, however, this configuration displays the code description from the *pickListDataSet*. So if the code description is changed in your *pickListDataSet*, that change will be reflected in your *DataSet* since this display value is always “looked up”. In the first configuration described above, if the code description changes in the *pickListDataSet*, your *DataSet* will still show the old “fill in” values.

The lookup column is a special calculated column that retrieves its display values from the *pickListDataSet*. Although the *pickList* editor is activated when edits are attempted on this column, no value can be set on this column. It always retrieves its values by looking them up in the *pickListDataSet*. The lookup operation uses the same mechanism and properties described in the second configuration above. To create a lookup *Column*, you must add a *Column* to your *DataSet* and set its *calcType* property to *CalcType.LOOKUP*. Then set the *pickList* property much like configuration #2. The only difference is that the lookup column cannot be one of the columns listed in the *destinationColumns* property.

PickListDescriptor constructors

PickListDescriptor(com.borland.dx.dataset.DataSet, java.lang.String[], java.lang.String[], java.lang.String[], boolean)

```
public PickListDescriptor(DataSet pickListDataSet, String[] pickListColumns, String[]
    pickListDisplayColumns, String[] destinationColumns, boolean enforceIntegrity)
```

Constructs a *PickListDescriptor* object with the properties stated in its parameters.

<i>pickListDataSet</i>	The <i>DataSet</i> object that contains the data for display in the pick list.
<i>pickListColumns</i>	The columns of the <i>pickListDataSet</i> from which values in the selected row are copied to the <i>destinationColumns</i> .
<i>pickListDisplayColumns</i>	The <i>Column</i> components of the <i>DataSet</i> to display in the pick list. This property is expressed as an array of <i>String</i> column names. If your pick list contains multiple columns, you must also set the <i>itemEditor</i> property of the <i>Column</i> to an instance of the <i>PopupPickListItemEditor</i> . The default pick list item editor is the <i>PickListItemEditor</i> , which can only display a picklist with a single column.
<i>destinationColumns</i>	The <i>Column</i> components that are populated with the values associated with the selected pick list choice. This property is expressed as an array of <i>String</i> column names.
<i>enforceIntegrity</i>	Whether to enforce data integrity rules (data constraints) on the data added to the <i>destinationColumns</i> .

PickListDescriptor(com.borland.dx.dataset.DataSet, java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String, boolean)

```
public PickListDescriptor(DataSet pickListDataSet, String[] pickListColumns, String[]
    pickListDisplayColumns, String[] destinationColumns, String lookupDisplayColumn, boolean
    enforceIntegrity)
```

Constructs a *PickListDescriptor* object with the properties stated in its parameters.

<i>pickListDataSet</i>	The <i>DataSet</i> object that contains the data for display in the pick list.
<i>pickListColumns</i>	The columns of the <i>pickListDataSet</i> from which values in the selected row are copied to the <i>destinationColumns</i> .

<i>pickListDisplayColumns</i>	The <i>Column</i> components of the <i>DataSet</i> to display in the pick list. This property is expressed as an array of <i>String</i> names.
<i>destinationColumns</i>	The <i>Column</i> components that are populated with the values associated with the selected pick list choice. This property is expressed as an array of <i>String</i> column names.
<i>lookupDisplayColumn</i>	The name of the <i>Column</i> in the <i>picklistDataSet</i> to display the column whose <i>pickList</i> property is being set.
<i>enforceIntegrity</i>	Whether to enforce data integrity rules (data constraints) on the data added to the <i>destinationColumns</i> .

PickListDescriptor properties

Property	Implemented in
class*	java.lang.Object
destinationColumns*	this class
enforceIntegrity*	this class
lookupDisplayColumn*	this class
pickListColumns*	this class
pickListDataSet*	this class
pickListDisplayColumns*	this class

destinationColumns

public final String[] getDestinationColumns()

Read-only property that returns the *String* names of the *Column* components that are filled in when an entry in the pick list is selected. Set this property using the *PickListDescriptor* constructor. This property is also used for the display of lookup values and if not set, your application may generate a *java.lang.NullPointerException* error.

enforceIntegrity

public boolean isEnforceIntegrity()

Read-only property that returns whether to enforce data integrity rules (data constraints) on the data added to the *destinationColumns*. Set this property using the *PickListDescriptor* constructor. This property is not currently used.

lookupDisplayColumn

```
public final String getLookupDisplayColumn()
```

Read-only property that returns the name of the *Column* in the *picklistDataSet* to display for this (the column whose *pickList* is being defined) column in data-aware controls.

pickListColumns

```
public final String[] getPickListColumns()
```

Read-only property that returns the *Column* components to read from when filling in data from the *pickListDataSet* into the columns specified in the *destinationColumns* property. Set this property using the *PickListDescriptor* constructor.

If your pick list contains multiple columns, you must also set the *itemEditor* property of the *Column* to an instance of the *PopupPickListItemEditor*. The default pick list *item editor* is the *PickListItemEditor*, which can only display picklist with a single column.

pickListDataSet

```
public final DataSet getPickListDataSet()
```

Read-only property that returns the *pickListDataSet* used to choose values from. Set this property using the *PickListDescriptor* constructor.

pickListDisplayColumns

```
public final String[] getPickListDisplayColumns()
```

Read-only property that returns a *String* array of the *Column* names to display from the *pickListDataSet*. Set this property using the *PickListDescriptor* constructor.

PickListDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object

Method	Implemented in
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

Provider class (abstract)

dx.dataset package

- Extends** java.lang.Object
- Extended by** com.borland.dx.sql.dataset.DataModelProvider, com.borland.dx.sql.dataset.JdbcProvider
- Implements** com.borland.dx.dataset.Designable, java.io.Serializable

The *Provider* class is an abstract base class that “provides” (or populates) a *DataSet* with data. Extend this class if you want to create a custom provider. The instantiable subclasses (technically, subclasses of its *JDBCProvider* subclass) *QueryProvider* and *ProcedureProvider* are included. These classes collect provider functionality using queries and stored procedures on JDBC data sources.

Provider properties

Property	Implemented in
accumulateResults*	this class
class*	java.lang.Object
parameterRow	this class

accumulateResults

public boolean isAccumulateResults()

Return **false** if new data provide requests should empty the associated *StorageDataSet*. Return **true** if new data provide requests should leave existing rows in the associated *StorageDataSet*.

parameterRow

public ReadWriteRow getParameterRow()
public void setParameterRow(ReadWriteRow value)

This is the *parameterRow* that will be used by extensions of *StorageDataSet*, like *QueryDataSet* and *ProcedureDataSet*, to fill in parameter values for parameterized queries or stored procedures. If a *TableDataSet* extension of *StorageDataSet* has a *QueryProvider* or *ProcedureProvider*, this property will

also be used by those providers to fill in parameter values for parameterized queries or stored procedures.

Provider methods

Method	Implemented in
<code>checkIfBusy(com.borland.dx.dataset.StorageDataSet)</code>	this class
<code>checkMasterLink(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.MasterLinkDescriptor)</code>	this class
<code>clone()</code>	java.lang.Object
<code>close(com.borland.dx.dataset.StorageDataSet, boolean)</code>	this class
<code>equals(java.lang.Object)</code>	java.lang.Object
<code>finalize()</code>	java.lang.Object
<code>hashCode()</code>	java.lang.Object
<code>hasMoreData(com.borland.dx.dataset.StorageDataSet)</code>	this class
<code>notify()</code>	java.lang.Object
<code>notifyAll()</code>	java.lang.Object
<code>provideData(com.borland.dx.dataset.StorageDataSet, boolean)</code>	this class
<code>provideMoreData(com.borland.dx.dataset.StorageDataSet)</code>	this class
<code>toString()</code>	java.lang.Object
<code>wait()</code>	java.lang.Object
<code>wait(long, int)</code>	java.lang.Object
<code>wait(long)</code>	java.lang.Object

checkIfBusy(com.borland.dx.dataset.StorageDataSet)

```
public void checkIfBusy(StorageDataSet dataSet)
```

Some implementations of the *provideData* method may optionally provide the data asynchronously. A *StorageDataSet* has to block actions such as resolving until the asynchronous data is present. This method allows an implementation to give an appropriate error message by raising a *DataSetException*. The default action is to do nothing, i.e. no asynchronous providing.

checkMasterLink(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.MasterLinkDescriptor)

```
public void checkMasterLink(StorageDataSet dataSet, MasterLinkDescriptor masterLink)
```

Called to validate the *masterLink* property. When the *MasterLinkDescriptor's* *fetchAsNeeded* property is enabled (**true**), the *QueryProvider* uses this method to check if there is a WHERE clause in the query. If no WHERE clause is specified, the *QueryProvider* throws a *DataSetException*.

close(com.borland.dx.dataset.StorageDataSet, boolean)

```
public void close(StorageDataSet dataSet, boolean loadRemainingRows)
```

Releases resources kept for loading data on demand. *StorageDataSet* calls this method when the storage is being closed and when calling *StorageDataSet.closeProvider*. The *loadRemainingRows* parameter controls whether the rest of the data (if more data is available) is loaded or not.

hasMoreData(com.borland.dx.dataset.StorageDataSet)

```
public boolean hasMoreData(StorageDataSet dataSet)
```

Some implementations of a *Provider* may allow to provide part of the data, then load more data on demand. This method should return **true** if there is more data to be loaded. To load the data, call the *provideMoreData* method.

Note *StorageDataSet* will attempt to retrieve more data in the following three cases:

- 1 Navigation with *dataSet.next()* goes past the last record.
- 2 *dataSet.last()* is called.
- 3 Whenever any UI component gets close to the last record.

Then, *Provider.hasMoreData()* is called. If it returns **true**, *Provider.provideMoreData()* is called.

provideData(com.borland.dx.dataset.StorageDataSet, boolean)

```
public abstract void provideData(StorageDataSet dataSet, boolean toOpen)
```

Provides the data for a *DataSet*. The source of the data, and the method of retrieving the data is up to the implementation of this abstract method. The *toOpen* parameter indicates whether this method is called as part of opening this *StorageDataSet*.

provideMoreData(com.borland.dx.dataset.StorageDataSet)

```
public void provideMoreData(StorageDataSet dataSet)
```

Some implementations of a *Provider* may allow to provide part of the data and then load more data on demand. This method provides more data if there is more data to be loaded. If no more data is available, this method simply returns.

Note *StorageDataSet* will attempt to retrieve more data in the following three cases:

- 1 Navigation with *dataSet.next()* goes past the last record.
- 2 *dataSet.last()* is called.
- 3 Whenever any UI component gets close to the last record.

Then, *Provider.hasMoreData()* is called. If it returns **true**, *Provider.provideMoreData()* is called.

ProviderHelp class

dx.dataset package

Extends java.lang.Object

This class collects utility functions that operate on the *StorageDataSet* and *DataSet* classes that are helpful for provider/resolver implementations. These methods are implemented inside *StorageDataSet*, but are not functions typical for *DataSet* usage.

ProviderHelp properties

Property	Implemented in
class*	java.lang.Object

ProviderHelp methods

Method	Implemented in
clone()	java.lang.Object
endResolution(com.borland.dx.dataset.StorageDataSet)	this class
equals(java.lang.Object)	java.lang.Object
failIfOpen(com.borland.dx.dataset.StorageDataSet)	this class
finalize()	java.lang.Object
getResolverDataSet(com.borland.dx.dataset.DataSet)	this class
getStructureAge(com.borland.dx.dataset.StorageDataSet)	this class
hashCode()	java.lang.Object
initData(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column[], boolean, boolean, boolean)	this class
initData(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column[], boolean, boolean)	this class
isCopyProviderStreams(com.borland.dx.dataset.StorageDataSet)	this class
isProviderPropertyChanged (com.borland.dx.dataset.StorageDataSet)	this class
markPendingStatus(com.borland.dx.dataset.DataSet, boolean)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
setMetaDataMissing(com.borland.dx.dataset.StorageDataSet, boolean)	this class
setProviderPropertyChanged(com.borland.dx.dataset.StorageDataSet, boolean)	this class
startResolution(com.borland.dx.dataset.StorageDataSet, boolean)	this class
toString()	java.lang.Object

Method	Implemented in
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

endResolution(com.borland.dx.dataset.StorageDataSet)

public static final void endResolution(StorageDataSet dataSet)

Used by the *ResolutionManager* to signal that *StorageDataSet* is no longer in a “resolving” mode. Once the *startResolution()* method is called, any other providing or resolving operation is prohibited until this method is called.

faillfOpen(com.borland.dx.dataset.StorageDataSet)

public static final void faillfOpen(StorageDataSet dataSet)

Causes a *DataSetException* to be thrown if the *DataSet* is open.

getResolverDataSet(com.borland.dx.dataset.DataSet)

public static final StorageDataSet getResolverDataSet(DataSet dataSet)

Returns a *StorageDataSet* that contains the metadata for a given *StorageDataSet* or *DataSetView*.

getStructureAge(com.borland.dx.dataset.StorageDataSet)

public static final int getStructureAge(StorageDataSet dataSet)

Returns an int, which can be saved and used by a provider as a flag that a column restructure has occurred. This number is incremented each time a column structure changes. Therefore, if a resolver initially calls this method and stores the return value, any change in its return value on a subsequent call indicates that the resolver must discard any cached information about the *DataSet*.

initData(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column[], boolean, boolean)

public static final int[] initData(StorageDataSet dataSet, Column[] columns, boolean updateColumns, boolean keepExistingColumns)

Used by providers, *initData()* initializes the data storage of a *DataSet* for a new set of columns and returns an ordinal map of the passed in columns to the corresponding columns in the *DataSet*. This method differs from the *StorageDataSet.setColumns()* method in that it preserves persistent columns. Note that several column properties in the columns array will be merged in with existing columns in the *StorageDataSet* columns that have the same name.

The `RMIPProvider.java` file in the `samples/com/borland/samples/DataExpress/datasetdata` directory of your JBuilder installation provides an example.

<i>dataSet</i>	The <i>StorageDataSet</i> whose data storage is to be initialized.
<i>columns</i>	The array of columns that represents the added columns.
<i>updateColumns</i>	Boolean that determines whether columns will be merged into existing persistent columns. If true, the columns will not be added. Instead the returned column map is computed with the current columns. If both <i>updateColumns</i> and <i>keepExistingColumns</i> parameters are true, non-persistent columns will also be retained.
<i>keepExistingColumns</i>	If true, new columns will be merged into existing columns. If false all non-persistent columns are removed before merging in the persistent columns.

initData(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column[], boolean, boolean, boolean)

```
public static final int[] initData(StorageDataSet dataSet, Column[] columns, boolean updateColumns,
    boolean keepExistingColumns, boolean emptyRows)
```

This method has been deprecated. Use *initData(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column[], boolean, boolean)* instead.

isCopyProviderStreams(com.borland.dx.dataset.StorageDataSet)

```
public static final boolean isCopyProviderStreams(StorageDataSet dataSet)
```

A provider can call this method to determine if it should make copies of data from columns of type *Variant.INPUT_STREAM*. The method will return **false** if the storage is going to copy the stream, thus there is no need for the provider to copy the data. The method returns **true** if the storage is just storing a reference to the *InputStream*. In this case, the provider might need to make a memory copy of the data, if the stream has a limited lifetime or is not resetable.

isProviderPropertyChanged(com.borland.dx.dataset.StorageDataSet)

```
public static final boolean isProviderPropertyChanged(StorageDataSet dataSet)
```

Reflects whether this *StorageDataSet* has received some property change which could affect the column structure or set of row data, for example, a change in *QueryDescriptor* or *TextDataFile*. Used by *Provider* classes. The

JdbcProvider uses this to decide if it needs to discard cached information about the JDBC data source.

See also *setProviderPropertyChanged(StorageDataSet,boolean)*

markPendingStatus(com.borland.dx.dataset.DataSet, boolean)

public static final void markPendingStatus(DataSet dataSet, boolean on)

Marks a row as pending resolution. Used by the *ResolutionManager* when saving changes from a *DataSet* to a remote data provider that supports transactions, for example, JDBC connections.

setMetaDataMissing(com.borland.dx.dataset.StorageDataSet, boolean)

public static final void setMetaDataMissing(StorageDataSet dataSet, boolean hasRowIds)

Normally set by *Providers* to indicate that there is insufficient metadata to post any changes back to its original source.

setProviderPropertyChanged(com.borland.dx.dataset.StorageDataSet, boolean)

public static final void setProviderPropertyChanged(StorageDataSet dataSet, boolean propertiesChanged)

Used by *Provider* classes. This property reflects whether this *StorageDataSet* has received some property change which could affect the column structure or set of row data, for example, a change in *QueryDescriptor* or *TextDataFile*.

See also *isProviderPropertyChanged(com.borland.dx.dataset.StorageDataSet)*

startResolution(com.borland.dx.dataset.StorageDataSet, boolean)

public static final void startResolution(StorageDataSet dataSet, boolean postEdits)

Used by the *ResolutionManager* to place a *StorageDataSet* into a “resolving” mode. This mode prohibits any other resolution or providing to occur. After the resolution phase is complete, the *endResolution()* method should be called. If the *postEdits* parameter is **true**, the *StorageDataSet* will post an unposted row for itself and any *DataSetViews* that may be associated with the *StorageDataSet*.

ReadRow class (abstract)

dx.dataset package

Extends java.lang.Object

Extended by com.borland.dx.dataset.ReadWriteRow,
com.borland.dx.dataset.InternalRow

Implements java.io.Serializable

The *ReadRow* class provides read access to a row of data. It has methods to read values from a single *Column* according to its data type, as well as methods to read the value from a *Column* of any data type into a *Variant*. It also has methods to compare or copy an entire row or a subset of its columns to another row.

The *ReadRow* class is extended by *ReadWriteRow*, which provides similar methods to write values to *Columns*. The *ReadWriteRow* class is in turn extended by *DataSet*, *DataRow*, and *ParameterRow*. These three classes all use the read and write methods in *ReadRow* and *ReadWriteRow* heavily to manipulate *Column* values.

You can use the *equals()*, *findDifference()*, and *copyTo()* methods to compare two rows from the same or different data sets, or to copy rows from one data set to another.

ReadRow properties

Property	Implemented in
class*	java.lang.Object
columnCount*	this class
columns*	this class

columnCount

public final int getColumnCount()

Read-only property that returns the count of *Column* components.

columns

public Column[] getColumns()

Read-only property that returns an array of columns.

ReadRow methods

Method	Implemented in
clone()	java.lang.Object
copyTo(com.borland.dx.dataset.ReadWriteRow)	this class
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	this class
equals(com.borland.dx.dataset.ReadRow)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	this class
findModified(int)	this class
findOrdinal(java.lang.String)	this class
format(int)	this class
format(java.lang.String)	this class
getArrayLength(java.lang.String)	this class
getBigDecimal(int)	this class
getBigDecimal(java.lang.String)	this class
getBinaryStream(int)	this class
getBoolean(int)	this class
getBoolean(java.lang.String)	this class
getByte(int)	this class
getByte(java.lang.String)	this class
getByteArray(int)	this class
getByteArray(java.lang.String)	this class
getColumn(int)	this class
getColumn(java.lang.String)	this class
getColumnNames(int)	this class
getDate(int)	this class
getDate(java.lang.String)	this class
getDouble(int)	this class
getDouble(java.lang.String)	this class
getFloat(int)	this class
getFloat(java.lang.String)	this class
getInputStream(int)	this class
getInputStream(java.lang.String)	this class
getInt(int)	this class
getInt(java.lang.String)	this class
getLong(int)	this class
getLong(java.lang.String)	this class
getObject(int)	this class

Method	Implemented in
<code>getObject(java.lang.String)</code>	this class
<code>getShort(int)</code>	this class
<code>getShort(java.lang.String)</code>	this class
<code>getString(int)</code>	this class
<code>getString(java.lang.String)</code>	this class
<code>getTime(int)</code>	this class
<code>getTime(java.lang.String)</code>	this class
<code>getTimestamp(int)</code>	this class
<code>getTimestamp(java.lang.String)</code>	this class
<code>getVariant(int, com.borland.dx.dataset.Variant)</code>	this class
<code>getVariant(java.lang.String, com.borland.dx.dataset.Variant)</code>	this class
<code>hasColumn(java.lang.String)</code>	this class
<code>hashCode()</code>	java.lang.Object
<code>isAssignedNull(int)</code>	this class
<code>isAssignedNull(java.lang.String)</code>	this class
<code>isCompatibleList(com.borland.dx.dataset.ReadRow)</code>	this class
<code>isModified(int)</code>	this class
<code>isModified(java.lang.String)</code>	this class
<code>isNull(int)</code>	this class
<code>isNull(java.lang.String)</code>	this class
<code>isUnassignedNull(int)</code>	this class
<code>isUnassignedNull(java.lang.String)</code>	this class
<code>notify()</code>	java.lang.Object
<code>notifyAll()</code>	java.lang.Object
<code>toString()</code>	this class
<code>wait()</code>	java.lang.Object
<code>wait(long, int)</code>	java.lang.Object
<code>wait(long)</code>	java.lang.Object

copyTo(com.borland.dx.dataset.ReadWriteRow)

```
public void copyTo(ReadWriteRow destRow)
```

Copies the row values from this row to *destRow*. The copy is not performed for destination *Columns* that are *readOnly*; no *Exception* is generated. If the *Column* components of the *destRow* are not from the same *DataSet*, columns with the same name are copied, assuming the data types of the columns match. If this *ReadRow* does not have columns with the same type and name as all columns in *destRow*, a *DataSetException* is thrown.

Call this method when this *ReadRow* and the destination *destRow* are identical, or when the columns of the *ReadRow* are a subset of those in the *destRow*. If the structure of this *ReadRow* component is not similar to that of

the *destRow*, call the *copyTo(String[], ReadRow, String[], ReadWriteRow)* method instead.

copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)

```
public static void copyTo(String[] sourceNames, ReadRow sourceRow, String[] destNames,
    ReadWriteRow destRow)
```

Copies values of the *ReadRow* to a *ReadWriteRow*, given an array of source and destination names. If a destination *Column* is *readOnly*, the copy is not performed for that column; no *Exception* is generated. Use this method when the structure of this *ReadRow* is not identical to that of the *destRow*, to specify which *Columns* of the *ReadRow* get copied to which *Columns* in the *destRow*.

equals(com.borland.dx.dataset.ReadRow)

```
public final boolean equals(ReadRow compareRow)
```

Returns **true** if the values of columns in *compareRow* are equal to the column values in this row that have the same name. The *compareRow* parameter may be a scoped row containing a subset of the columns in this row. If this row does not have columns with the same name and type as all columns in *compareRow*, a *DataSetException* is thrown.

findDifference(int, com.borland.dx.dataset.ReadRow)

```
public final int findDifference(int startOrdinal, ReadRow compareRow)
```

Returns the ordinal of the first column value that differs between this row and *compareRow* starting from *startOrdinal*. If there are no more differences, -1 is returned. On error, this method throws a *DataSetException*.

findModified(int)

```
public final int findModified(int startOrdinal)
```

Returns the ordinal of the first column value that has been modified. If there are no more modified columns after *startOrdinal*, -1 is returned.

findOrdinal(java.lang.String)

```
public final int findOrdinal(String columnName)
```

Returns the ordinal of the column specified in *columnName*. This method is slightly more performant than *hasColumn* if there is a good chance the column does not exist.

format(int)

public final String format(int ordinal)

Returns the *String* representation of the value at the ordinal position using a *Column* formatter. On error, this method throws a *DataSetException*.

format(java.lang.String)

public final String format(String columnName)

Returns a *String* representation of the value at the specified column name, using a *Column* formatter.

getArrayLength(java.lang.String)

public final int getArrayLength(String columnName)

Returns the length of the BYTE_ARRAY. On error, this method throws a *DataSetException*.

getBigDecimal(int)

public final BigDecimal getBigDecimal(int ordinal)

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *BigDecimal*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.BIGDECIMAL*.

See also [getBigDecimal\(java.lang.String\)](#)

getBigDecimal(java.lang.String)

public final BigDecimal getBigDecimal(String columnName)

Returns the value in the *Column* named *columnName* as a *BigDecimal*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.BIGDECIMAL*.

This method is typically preferred over *getBigDecimal(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions. Note that ordinal access can be slightly faster, especially for *DataSets* with more than 20 columns.

getBinaryStream(int)

public final InputStream getBinaryStream(int ordinal)

This method has been deprecated. Use *getInputStream(int)*.

getBoolean(int)

```
public final boolean getBoolean(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a boolean. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.BOOLEAN*.

See also `getBoolean(java.lang.String)`

getBoolean(java.lang.String)

```
public final boolean getBoolean(String columnName)
```

Returns the value in the *Column* named *columnName* as a boolean. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.BOOLEAN*.

This method is typically preferred over *getBoolean(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getByte(int)

```
public final byte getByte(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a byte. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.BYTE*.

See also `getByte(java.lang.String)`

getByte(java.lang.String)

```
public final byte getByte(String columnName)
```

Returns the value in the *Column* named *columnName* as a byte. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.BYTE*.

This method is typically preferred over *getByte(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getByteArray(int)

```
public final byte[] getByteArray(int ordinal)
```

This method is used internally by other *com.borland* classes. You should never use this method directly.

getByteArray(java.lang.String)

```
public final byte[] getByteArray(String columnName)
```

This method is used internally by other *com.borland* classes. You should never use this method directly.

getColumn(int)

```
public final Column getColumn(int ordinal)
```

Returns the *Column* component at the specified ordinal index location.

getColumn(java.lang.String)

```
public final Column getColumn(String columnName)
```

Returns the *Column* component for the specified *columnName*. Similar to *hasColumn(java.lang.String)*, however this method throws a *DataSetException* if the *Column* is not found.

getColumnNames(int)

```
public final String[] getColumnNames(int columnCount)
```

Returns an array containing the names of the first *columnCount* Columns. On error, this method throws a *DataSetException*.

getDate(int)

```
public final java.sql.Date getDate(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *Date*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.DATE*.

See also [*getDate\(java.lang.String\)*](#)

getDate(java.lang.String)

```
public final java.sql.Date getDate(String columnName)
```

Returns the value in the *Column* named *columnName* as a *Date*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.DATE*.

This method is typically preferred over *getDate(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns (columns that are automatically added to a query to provide a unique row identifier) and other conditions.

getDouble(int)

```
public final double getDouble(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a double. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.DOUBLE*.

See also `getDouble(java.lang.String)`

getDouble(java.lang.String)

```
public final double getDouble(String columnName)
```

Returns the value in the *Column* named *columnName* as a double. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.DOUBLE*.

This method is typically preferred over *getDouble(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getFloat(int)

```
public final float getFloat(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a float. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.FLOAT*.

See also `getFloat(java.lang.String)`

getFloat(java.lang.String)

```
public final float getFloat(String columnName)
```

Returns the value in the *Column* named *columnName* as a float. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.FLOAT*.

This method is typically preferred over *getFloat(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getInputStream(int)

```
public final InputStream getInputStream(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as an *InputStream*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.INPUTSTREAM*.

See also `getInputStream(java.lang.String)`

getInputStream(java.lang.String)

```
public final InputStream getInputStream(String columnName)
```

Returns the value in the *Column* named *columnName* as an *InputStream*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.INPUTSTREAM*.

This method is typically preferred over *getInputStream(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getInt(int)

```
public final int getInt(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as an int. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.INT*.

See also `getInt(java.lang.String)`

getInt(java.lang.String)

```
public final int getInt(String columnName)
```

Returns the value in the *Column* named *columnName* as an int. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not an *Variant.INT*.

This method is typically preferred over *getInt(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getLong(int)

```
public final long getLong(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a long. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.LONG*.

See also `getLong(java.lang.String)`

getLong(java.lang.String)

```
public final long getLong(String columnName)
```

Returns the value in the *Column* named *columnName* as a long. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.LONG*.

This method is typically preferred over *getLong(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getObject(int)

```
public final Object getObject(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as an *Object*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.OBJECT*.

See also [getObject\(java.lang.String\)](#)

getObject(java.lang.String)

```
public final Object getObject(String columnName)
```

Returns the value in the *Column* named *columnName* as an *Object*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.OBJECT*.

This method is typically preferred over *getObject(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getShort(int)

```
public final short getShort(int ordinal)
```

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a short. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.SHORT*.

See also [getShort\(java.lang.String\)](#)

getShort(java.lang.String)

```
public final short getShort(String columnName)
```

Returns the value in the *Column* named *columnName* as a short. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.SHORT*.

This method is typically preferred over *getShort(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getString(int)

public final String getString(int ordinal)

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *String*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.STRING*.

See also *getString(java.lang.String)*

getString(java.lang.String)

public final String getString(String columnName)

Returns the value in the *Column* named *columnName* as a *String*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.STRING*.

This method is typically preferred over *getString(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getTime(int)

public final Time getTime(int ordinal)

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *Time*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.TIME*.

See also *getTime(java.lang.String)*

getTime(java.lang.String)

public final Time getTime(String columnName)

Returns the value in the *Column* named *columnName* as a *Time*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.TIME*.

This method is typically preferred over *getTime(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getTimestamp(int)

public final Timestamp getTimestamp(int ordinal)

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *Timestamp*. A *DataSetException* is thrown if the ordinal position *ordinal* does not exist, or if the column's data type is not *Variant.TIMESTAMP*.

See also [getTimestamp\(java.lang.String\)](#)

getTimestamp(java.lang.String)

public final Timestamp getTimestamp(String columnName)

Returns the value in the *Column* named *columnName* as a *Timestamp*. A *DataSetException* is thrown if *columnName* does not exist, or if the column's data type is not *Variant.TIMESTAMP*.

This method is typically preferred over *getTimestamp(int)* since it is more reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

getVariant(int, com.borland.dx.dataset.Variant)

public void getVariant(int ordinal, Variant value)

Returns the value in the *Column* indicated by its ordinal position in the *ReadRow* as a *Variant*. The *Variant* is returned as the *value* parameter passed into this method.

getVariant(java.lang.String, com.borland.dx.dataset.Variant)

public void getVariant(String columnName, Variant value)

Returns the value in the *Column* named *columnName* as a *Variant*. The *Variant* is returned as the *value* parameter passed into this method.

hasColumn(java.lang.String)

public final Column hasColumn(String columnName)

Returns the *Column* object as specified by its String name. Similar to the *getColumn* method, however this method returns null if the *Column* is not found instead of throwing a *DataSetException*.

isAssignedNull(int)

public final boolean isAssignedNull(int ordinal)

Determines whether the data value at location *ordinal* is an assigned null value. If it returns **true**, the value is an assigned null value; otherwise, it is not and returns **false**.

isAssignedNull(java.lang.String)

public final boolean isAssignedNull(String columnName)

Determines whether the data value identified by its column name is an assigned null value. If it returns **true**, the value is an assigned null value; otherwise, it is not and returns **false**.

isCompatibleList(com.borland.dx.dataset.ReadRow)

public final boolean isCompatibleList(ReadRow row)

This method is used internally by other *com.borland* classes. You should never use this method directly.

isModified(int)

public boolean isModified(int ordinal)

Returns **true** if the value at the specified *ordinal* has been modified.

isModified(java.lang.String)

public boolean isModified(String columnName)

Returns **true** if the value at the specified *columnName* has been modified.

isNull(int)

public final boolean isNull(int ordinal)

Returns **true** if the value at the specified *ordinal* is either an assigned or unassigned null; **false** otherwise. To determine if the value is an unassigned null, call the *isUnassignedNull(int)* method.

isNull(java.lang.String)

public final boolean isNull(String columnName)

Returns **true** if the value at the specified *columnName* is either an assigned or unassigned null; **false** otherwise. To determine if the value is an unassigned null, call the *isUnassignedNull(String)* method.

isUnassignedNull(int)

public final boolean isUnassignedNull(int ordinal)

Returns **true** if the value at the specified *ordinal* is an unassigned null; **false** otherwise.

isUnassignedNull(java.lang.String)

public final boolean isUnassignedNull(String columnName)

Returns **true** if the value at the specified *columnName* is an unassigned null; **false** otherwise.

toString()

public String toString()

Returns a *String* representation of this *ReadRow* object.

Overrides java.lang.Object.toString()

ReadWriteRow class (abstract)

dx.dataset package

Extends com.borland.dx.dataset.ReadRow

Extended by com.borland.dx.dataset.DataRow, com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ParameterRow, com.borland.dx.dataset.RowIterator

Implements java.io.Serializable

The *ReadWriteRow* class adds write access to its superclass *ReadRow*. It adds behavior to write values to a single *Column* according to its data type, as well as methods to write the value from a *Column* of any data type into a *Variant*. It also has methods to clear values.

The *ReadWriteRow* is an abstract class that is extended by *DataSet*, *DataRow*, and *ParameterRow*. These three classes all use the read and write methods in *ReadWriteRow* and its superclass *ReadRow* heavily to manipulate data values.

ReadWriteRow properties

Property	Implemented in
assignedNull**	this class
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow

Property	Implemented in
columns*	com.borland.dx.dataset.ReadRow
unassignedNull**	this class

assignedNull

```
public final void setAssignedNull(String columnName)
public final void setAssignedNull(int ordinal)
```

Write-only property that explicitly sets a *Column* to **null** (as opposed to a value that is simply not assigned). The *Column* can be specified by name or by its ordinal position in the *ReadWriteRow*, though the column name is preferred because the ordinal position may change unexpectedly. The read accessors for this property, *isAssignedNull(int)* and *isAssignedNull(String)* are implemented in *com.borland.dx.dataset.ReadRow*.

The JDBC *QueryResolver*, used by *QueryDataSet* to save changes, will use a “set null” clause for columns with assigned null values.

unassignedNull

```
public final void setUnassignedNull(String columnName)
public final void setUnassignedNull(int ordinal)
```

Write-only property that sets a *Column* to an unassigned null value, which represents a value that was never assigned. The *Column* can be specified by name or by its ordinal position in the *ReadWriteRow*, though the column name is preferred because the ordinal position may change unexpectedly. The read accessors for this property, *isUnassignedNull(int)* and *isUnassignedNull(java.lang.String)* are implemented in *com.borland.dx.dataset.ReadRow*.

The JDBC *QueryResolver* does not specify columns with unassigned null values in the INSERT statements submitted to the JDBC driver. This allows the JDBC data source to fill in any server-side defaults for that column.

ReadWriteRow methods

Method	Implemented in
clearValues()	this class
clone()	java.lang.Object
copyTo(com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow

Method	Implemented in
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>findDifference(int, com.borland.dx.dataset.ReadRow)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>findModified(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>findOrdinal(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>format(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>format(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getArrayLength(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getBigDecimal(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getBigDecimal(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getBinaryStream(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getBoolean(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getBoolean(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getByte(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getByte(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getByteArray(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getByteArray(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getColumn(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getColumn(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getColumnNames(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getDate(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getDate(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getDouble(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getDouble(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getFloat(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getFloat(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getInputStream(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getInputStream(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getInt(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getInt(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getLong(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getLong(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getObject(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getObject(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getShort(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getShort(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getString(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getString(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getTime(int)</code>	<code>com.borland.dx.dataset.ReadRow</code>
<code>getTime(java.lang.String)</code>	<code>com.borland.dx.dataset.ReadRow</code>

Method	Implemented in
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hashCode()	java.lang.Object
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.ReadRow
isModified(java.lang.String)	com.borland.dx.dataset.ReadRow
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
notify()	java.lang.Object
notifyAll()	java.lang.Object
requiredColumnsCheck()	this class
setBigDecimal(int, java.math.BigDecimal)	this class
setBigDecimal(java.lang.String, java.math.BigDecimal)	this class
setBoolean(int, boolean)	this class
setBoolean(java.lang.String, boolean)	this class
setByte(int, byte)	this class
setByte(java.lang.String, byte)	this class
setByteArray(int, byte[], int)	this class
setByteArray(java.lang.String, byte[], int)	this class
setDate(int, java.sql.Date)	this class
setDate(int, long)	this class
setDate(java.lang.String, java.sql.Date)	this class
setDate(java.lang.String, long)	this class
setDefaultValues()	this class
setDouble(int, double)	this class
setDouble(java.lang.String, double)	this class
setFloat(int, float)	this class
setFloat(java.lang.String, float)	this class
setInputStream(int, java.io.InputStream)	this class
setInputStream(java.lang.String, java.io.InputStream)	this class
setInt(int, int)	this class

Method	Implemented in
setInt(java.lang.String, int)	this class
setLong(int, long)	this class
setLong(java.lang.String, long)	this class
setObject(int, java.lang.Object)	this class
setObject(java.lang.String, java.lang.Object)	this class
setShort(int, short)	this class
setShort(java.lang.String, short)	this class
setString(int, java.lang.String)	this class
setString(java.lang.String, java.lang.String)	this class
setTime(int, java.sql.Time)	this class
setTime(int, long)	this class
setTime(java.lang.String, java.sql.Time)	this class
setTime(java.lang.String, long)	this class
setTimestamp(int, java.sql.Timestamp)	this class
setTimestamp(int, long)	this class
setTimestamp(java.lang.String, java.sql.Timestamp)	this class
setTimestamp(java.lang.String, long)	this class
setVariant(int, com.borland.dx.dataset.Variant)	this class
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	this class
toString()	com.borland.dx.dataset.ReadRow
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

clearValues()

public final void clearValues()

Sets all values of the row to unassigned null. This method throws a *DataSetException* if any *Column* components have constraints such as *required*, or a minimum value.

The JDBC *QueryResolver* does not specify columns with unassigned null values in the INSERT statements submitted to the JDBC driver. This allows the JDBC data source to fill in any server-side defaults for that column.

requiredColumnsCheck()

public void requiredColumnsCheck()

Throws an *Exception* if any required columns have not been set to a non-null value.

setBigDecimal(int, java.math.BigDecimal)

public final void setBigDecimal(int ordinal, BigDecimal value)

Sets the *Column* indicated by its ordinal position to *value*. A *DataSetException* is thrown if the position ordinal does not exist, if the data type of the *value* parameter is not *BigDecimal*, or if the column's type is not *Variant.BIGDECIMAL*.

See also `setBigDecimal(java.lang.String, java.math.BigDecimal)`

setBigDecimal(java.lang.String, java.math.BigDecimal)

public final void setBigDecimal(String columnName, BigDecimal value)

Sets the *Column* indicated by *columnName* to *value*. A *DataSetException* is thrown if the data type of the *value* parameter is not *BigDecimal*, if the *columnName* does not exist, or if the column's type is not *Variant.BIGDECIMAL*.

This method is typically preferred over *setBigDecimal(int, BigDecimal)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions. Note that ordinal methods can be slightly faster, especially with *DataSets* that have more than 20 columns.

setBoolean(int, boolean)

public final void setBoolean(int ordinal, boolean value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not boolean, or the column's data type is not *Variant.BOOLEAN*.

See also `setBoolean(java.lang.String, boolean)`

setBoolean(java.lang.String, boolean)

public final void setBoolean(String columnName, boolean value)

Sets the *Column* indicated by *columnName* to *value*. A *DataSetException* is thrown if the data type of *value* is not boolean, the *columnName* does not exist, or the data type of the column is not *Variant.BOOLEAN*.

This method is typically preferred over *setBoolean(int,boolean)* because it does not involve the *Column*'s ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setByte(int, byte)

```
public final void setByte(int ordinal, byte value)
```

Sets the *Column* indicated by its *ordinal* position to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, if the data type of *value* is not byte, or if the column's data type is not *Variant.BYTE*.

See also *setByte(java.lang.String, byte)*

setByte(java.lang.String, byte)

```
public final void setByte(String columnName, byte value)
```

Sets the *Column* indicated by *columnName* to *value*. A *DataSetException* is thrown if the data type of *value* is not byte, the data type of the column is not *Variant.BYTE*, or the *columnName* does not exist.

This method is typically preferred over *setByte(int,byte)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setByteArray(int, byte[], int)

```
public final void setByteArray(int ordinal, byte[] value, int length)
```

This method is used internally by other *com.borland* classes. You should never use this method directly.

setByteArray(java.lang.String, byte[], int)

```
public final void setByteArray(String columnName, byte[] value, int length)
```

This method is used internally by other *com.borland* classes. You should never use this method directly.

setDate(int, java.sql.Date)

```
public final void setDate(int ordinal, java.sql.Date value)
```

Sets the *Column* indicated by its *ordinal* position to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *java.sql.Date*, or the data type of the *Column* is not *Variant.DATE*.

See also *setDate(java.lang.String, java.sql.Date)*

setDate(int, long)

```
public final void setDate(int ordinal, long value)
```

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January

1, 1970, 00:00:00 GMT. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not long, or the column's data type is not *Variant.DATE*.

See also *setDate(java.lang.String,long)*

setDate(java.lang.String, java.sql.Date)

```
public final void setDate(String columnName, java.sql.Date value)
```

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *java.sql.Date*, *columnName* does not exist, or the column's data type is not *Variant.DATE*.

This method is typically preferred over *setDate(int,Date)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setDate(java.lang.String, long)

```
public final void setDate(String columnName, long value)
```

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January 1, 1970, 00:00:00 GMT.

A *DataSetException* is thrown if the data type of *value* is not a long, the *columnName* does not exist, or the column data type is not *Variant.LONG*.

This method is typically preferred over *setDate(int,long)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setDefaultValues()

```
public void setDefaultValues()
```

Sets all values to column-specified default values.

setDouble(int, double)

```
public final void setDouble(int ordinal, double value)
```

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* as a double. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not double, or the column's data type is not *Variant.DOUBLE*.

See also *setDouble(java.lang.String, double)*

setDouble(java.lang.String, double)

```
public final void setDouble(String columnName, double value)
```

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not double, the *columnName* does not exist, or the column's data type is not *Variant.DOUBLE*.

This method is typically preferred over *setDouble(int,double)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setFloat(int, float)

```
public final void setFloat(int ordinal, float value)
```

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not float, or the data type of the *Column* is not *Variant.FLOAT*.

See also *setFloat(java.lang.String, float)*

setFloat(java.lang.String, float)

```
public final void setFloat(String columnName, float value)
```

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not float, the data type of the *Column* is not *Variant.FLOAT*, or if the *columnName* does not exist.

This method is typically preferred over *setFloat(int,float)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setInputStream(int, java.io.InputStream)

```
public final void setInputStream(int ordinal, InputStream value)
```

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *java.io.InputStream*, or the data type of the *Column* is not *Variant.INPUTSTREAM*.

See also *setInputStream(java.lang.String, java.io.InputStream)*

setInputStream([java.lang.String](#), [java.io.InputStream](#))

public final void setInputStream([String](#) columnName, [InputStream](#) value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not a *java.io.InputStream*, the data type of the *Column* is not *Variant.INPUTSTREAM*, or the *columnName* does not exist.

This method is typically preferred over *setInputStream(int,InputStream)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setInt([int](#), [int](#))

public final void setInt([int](#) ordinal, [int](#) value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *int*, or the column's data type is not *Variant.INT*.

See also [setInt\(\[java.lang.String\]\(#\), \[int\]\(#\)\)](#)

setInt([java.lang.String](#), [int](#))

public final void setInt([String](#) columnName, [int](#) value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *int*, the *columnName* is not found, or the data type of the *Column* is not *Variant.INT*.

This method is typically preferred over *setInt(int,int)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setLong([int](#), [long](#))

public final void setLong([int](#) ordinal, [long](#) value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *long*, or the data type of the *Column* is not *Variant.LONG*.

See also [setLong\(\[java.lang.String\]\(#\), \[long\]\(#\)\)](#)

setLong(java.lang.String, long)

public final void setLong(String columnName, long value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not long, the data type of the *Column* is not *Variant.LONG*, or the *columnName* does not exist.

This method is typically preferred over *setLong(int,long)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setObject(int, java.lang.Object)

public final void setObject(int ordinal, Object value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *Object*, or the data type of the *Column* is not *Variant.OBJECT*. An exception is thrown if the *Column.javaClass* property is set, and the value set is not the same class as *Column.javaClass*.

See also *setObject(java.lang.String, java.lang.Object)*

setObject(java.lang.String, java.lang.Object)

public final void setObject(String columnName, Object value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *java.lang.Object*, the data type of the *Column* is not *Variant.OBJECT*, or the *columnName* does not exist. An exception is thrown if the *Column.javaClass* property is set, and the value set is not the same class as *Column.javaClass*.

This method is typically preferred over *setObject(int,Object)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setShort(int, short)

public final void setShort(int ordinal, short value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not short, or the data type of the *Column* is not *Variant.SHORT*.

See also *setShort(java.lang.String, short)*

setShort(java.lang.String, short)

public final void setShort(String columnName, short value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not short, the data type of the *Column* is not *Variant.SHORT*, or the *columnName* does not exist.

This method is typically preferred over *setShort(int,short)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setString(int, java.lang.String)

public final void setString(int ordinal, String value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *java.lang.String*, or the data type of the *Column* is not *Variant.STRING*.

See also *setString(java.lang.String, java.lang.String)*

setString(java.lang.String, java.lang.String)

public final void setString(String columnName, String value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *String*, the data type of the *Column* is not *Variant.STRING*, or *columnName* does not exist.

This method is typically preferred over *setString(int,String)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setTime(int, java.sql.Time)

public final void setTime(int ordinal, Time value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *java.sql.Time*, or the data type of the *Column* is not *Variant.TIME*.

See also *setTime(java.lang.String, java.sql.Time)*

setTime(int, long)

public final void setTime(int ordinal, long value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January 1, 1970, 00:00:00 GMT. A *DataSetException* is thrown if the data type of *value* is not long, the data type of the *Column* is not *Variant.LONG*, or the specified *ordinal* is invalid.

See also `setTime(java.lang.String, java.sql.Time)`

setTime(java.lang.String, java.sql.Time)

public final void setTime(String columnName, Time value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *Time*, the data type of the *Column* is not *Variant.TIME*, or *columnName* does not exist.

This method is typically preferred over *setTime(int,Time)* or *setTime(int,long)* since these methods use the column's ordinal position. They are less reliable because the ordinal position may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setTime(java.lang.String, long)

public final void setTime(String columnName, long value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January 1, 1970, 00:00:00 GMT. A *DataSetException* is thrown if the data type of *value* is not long, the data type of the *Column* is not *Variant.TIME*, or the *columnName* does not exist.

This method is typically preferred over *setTime(int,Time)* or *setTime(int,long)* since these methods use the column's ordinal position. They are less reliable because the ordinal position may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setTimestamp(int, java.sql.Timestamp)

public final void setTimestamp(int ordinal, Timestamp value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not *java.sql.Timestamp*, or the data type of the *Column* is not *Variant.TIMESTAMP*.

See also `setTimestamp(java.lang.String, java.sql.Timestamp)`

setTimestamp(int, long)

public final void setTimestamp(int ordinal, long value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January 1, 1970, 00:00:00 GMT. A *DataSetException* is thrown if the *ordinal* position does not exist, the data type of *value* is not long, or the data type of the *Column* is not *Variant.TIMESTAMP*.

See also `setTimestamp(java.lang.String, long)`

setTimestamp(java.lang.String, java.sql.Timestamp)

public final void setTimestamp(String columnName, Timestamp value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the data type of *value* is not *java.sql.Timestamp*, the data type of the *Column* is not *Variant.TIMESTAMP*, or the *columnName* does not exist.

This method is typically preferred over `setTimestamp(int, Timestamp)` because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setTimestamp(java.lang.String, long)

public final void setTimestamp(String columnName, long value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. The *value* parameter is expressed in the number of milliseconds since January 1, 1970, 00:00:00 GMT. A *DataSetException* is thrown if the data type of *value* is not long, the *columnName* does not exist, or the data type of the *Column* is not *Variant.TIMESTAMP*.

This method is typically preferred over `setTimestamp(int, long)` because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

setVariant(int, com.borland.dx.dataset.Variant)

public final void setVariant(int ordinal, Variant value)

Sets the *Column* indicated by its *ordinal* position in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *ordinal* position does not exist, or if the data type of the value stored in the *Variant* (as returned by *Variant.getType()*) does not match with the *Column*'s data type (as returned by *Column.getDataType()*).

See also `setVariant(java.lang.String, com.borland.dx.dataset.Variant)`

setVariant(java.lang.String, com.borland.dx.dataset.Variant)

public final void setVariant(String columnName, Variant value)

Sets the *Column* indicated by *columnName* in the *ReadWriteRow* to *value*. A *DataSetException* is thrown if the *columnName* does not exist, or if the data type of the value stored in the *Variant* (as returned by *Variant.getType()*) does not match with the *Column*'s data type (as returned by *Column.getDataType()*).

This method is typically preferred over *setVariant(int,Variant)* because it does not involve the column's ordinal position, which is not always reliable. A column's ordinal value may unexpectedly change due to persistent columns, columns that are automatically added to a query to provide a unique row identifier, and other conditions.

Resolver class (abstract)

dx.dataset package

Extends	java.lang.Object
Extended by	com.borland.dx.sql.dataset.SQLResolver
Implements	com.borland.dx.dataset.Designable, java.io.Serializable

The *Resolver* class is a base class for *Resolver* objects such as *SQLResolver*. A *Resolver* should be able to update a snapshot of a data source with changes from another snapshot of a database. The changes might be performed on a row by row basis, as *SQLResolver* does, or in a batch mode.

Resolver properties

Property	Implemented in
class*	java.lang.Object

Resolver methods

Method	Implemented in
checkIfBusy(com.borland.dx.dataset.StorageDataSet)	this class
clone()	java.lang.Object
close(com.borland.dx.dataset.StorageDataSet)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object

Method	Implemented in
notify()	java.lang.Object
notifyAll()	java.lang.Object
resolveData(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

checkIfBusy(com.borland.dx.dataset.StorageDataSet)

public void checkIfBusy(StorageDataSet dataSet)

Some implementations of *resolveData(..)* may optionally provide the data asynchronously. A *StorageDataSet* has to block action such as editing and providing until the asynchronous provide is done. This method allows an implementation to give an appropriate error message by raising a *DataSetException*. The default action is to do nothing, i.e. no asynchronous providing.

close(com.borland.dx.dataset.StorageDataSet)

public void close(StorageDataSet dataSet)

Some implementations of *Resolver* cache information during the resolve. This method allows an implementation to release these resources and references for better garbage collection.

resolveData(com.borland.dx.dataset.DataSet)

public abstract void resolveData(DataSet dataSet)

Resolves the modified data in a *DataSet* back to the data source. The destination of the data, and the method of saving the data is up to the implementation of this abstract method.

ResolverAdapter class

dx.dataset package

Extends	java.lang.Object
Implements	com.borland.dx.dataset.ResolverListener, java.util.EventListener
This is an adapter class for <i>ResolverListener</i> , which is used as notification before and after a <i>StorageDataSet</i> is resolved.	

ResolverAdapter properties

Property	Implemented in
class*	java.lang.Object

ResolverAdapter methods

Method	Implemented in
clone()	java.lang.Object
deletedRow(com.borland.dx.dataset.ReadWriteRow)	this class
deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)	this class
deletingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ResolverResponse)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
insertedRow(com.borland.dx.dataset.ReadWriteRow)	this class
insertError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)	this class
insertingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ResolverResponse)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
updatedRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow)	this class
updateError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)	this class
updatingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ResolverResponse)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ResolverListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.ResolverAdapter

This interface is used as a notification before and after a *StorageDataSet* is resolved. This interface includes

- Methods that occur before a data set is resolved (methods containing “ing”). These events (*insertingRow*, *updatingRow*, *deletingRow*) can be used to validate the row being resolved, and prevent unwanted changes.
- Methods that occur after a data set is resolved (methods containing “ed”). These events (*insertedRow*, *updatedRow*, *deletedRow*) can be used to indicate that the action has been performed.
- Methods that occur when an exception is thrown in response to an attempt to resolve (methods ending in “Error”). The events (*insertError*, *updateError*, *deleteError*) can be used to trap errors during resolution, and take the appropriate action, such as aborting the transaction, ignoring the error, or retrying the resolution.

This listener is added to instances of the *QueryResolver* class. The *QueryResolver* is hooked to the *StorageDataSet* by setting the *resolver* property of the *StorageDataSet*.

ResolverListener methods

Method	Implemented in
<code>deletedRow(com.borland.dx.dataset.ReadWriteRow)</code>	this class
<code>deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>deletingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ResolverResponse)</code>	this class
<code>insertedRow(com.borland.dx.dataset.ReadWriteRow)</code>	this class
<code>insertError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)</code>	this class
<code>insertingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ResolverResponse)</code>	this class
<code>updatedRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow)</code>	this class

Method	Implemented in
updateError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)	this class
updatingRow(com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.ResolverResponse)	this class

deletedRow(com.borland.dx.dataset.ReadWriteRow)

public void deletedRow(ReadWriteRow row)

This method is called when the deletion of a row from the data set has been resolved on the server.

row The row that has been deleted.

deleteError(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow, com.borland.dx.dataset.DataSetException, com.borland.jb.util.ErrorResponse)

public void deleteError(DataSet dataSet, ReadWriteRow row, DataSetException ex, ErrorResponse response)

This method is called when an exception is thrown during resolution of a deletion from the *DataSet*.

- dataSet* The original *DataSet* passed in to be resolved. This can be used to position any controls bound to it if user interaction is needed.
- row* The row with the problem, positioned at the row that caused the error. This can be modified to correct the problem and retry the operation.
- ex* The exception that caused the error. Note that this may be a chained exception.
- response* Specify a response of *ABORT*, *IGNORE*, or *RETRY* for this error. These constants are defined in *util.ErrorResponse*. Note that an *ABORT* response causes all insert, update, and delete operations in the same transaction to be rolled back.

See also *com.borland.dx.dataset.DataSetException*

deletingRow([com.borland.dx.dataset.ReadWriteRow](#), [com.borland.dx.dataset.ResolverResponse](#))

public void deletingRow([ReadWriteRow](#) row, [ResolverResponse](#) response)

This method is called just before the deletion of a row from the data set is resolved on the server.

<i>row</i>	The row that is to be deleted.
<i>response</i>	Specify a response of <i>ABORT</i> , <i>IGNORE</i> , or <i>RETRY</i> for this error. These constants are defined in <i>util.ErrorResponse</i> . Note that an <i>ABORT</i> response causes all insert, update, and delete operations in the same transaction to be rolled back.

insertedRow([com.borland.dx.dataset.ReadWriteRow](#))

public void insertedRow([ReadWriteRow](#) row)

This method is called when the insertion of a row into the data set has been resolved on the server.

<i>row</i>	The row that has been inserted and resolved.
------------	--

insertError([com.borland.dx.dataset.DataSet](#), [com.borland.dx.dataset.ReadWriteRow](#), [com.borland.dx.dataset.DataSetException](#), [com.borland.jb.util.ErrorResponse](#))

public void insertError([DataSet](#) dataSet, [ReadWriteRow](#) row, [DataSetException](#) ex, [ErrorResponse](#) response)

This method is called when an exception is thrown during resolution of an insertion into the *DataSet*.

<i>dataSet</i>	The original <i>DataSet</i> passed in to be resolved, positioned at the row that caused the error. This can be used to position any controls bound to it if user interaction is needed.
<i>row</i>	The row with the problem. This can be modified to correct the problem and retry the operation.
<i>ex</i>	The exception that caused the error. Note that this may be a chained exception.
<i>response</i>	Specify a response of <i>ABORT</i> , <i>IGNORE</i> , or <i>RETRY</i> for this error. These constants are defined in <i>util.ErrorResponse</i> . Note that an <i>ABORT</i> response causes all insert, update, and delete operations in the same transaction to be rolled back.

See also [com.borland.dx.dataset.DataSetException](#)

insertingRow(`com.borland.dx.dataset.ReadWriteRow`, `com.borland.dx.dataset.ResolverResponse`)

public void insertingRow(`ReadWriteRow` row, `ResolverResponse` response)

This method is called when just before the insertion of a row into the data set is resolved to the server.

row The row that is about to be resolved.

response Specify a response of *ABORT*, *IGNORE*, or *RETRY* for this error. These constants are defined in *util.ErrorResponse*.

updatedRow(`com.borland.dx.dataset.ReadWriteRow`, `com.borland.dx.dataset.ReadRow`)

public void updatedRow(`ReadWriteRow` row, `ReadRow` oldRow)

This method is called when modifications to a row in the data set have been resolved on the server.

row The row that has been modified.

oldRow The original row.

updateError(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.ReadWriteRow`, `com.borland.dx.dataset.ReadRow`, `com.borland.dx.dataset.ReadWriteRow`, `com.borland.dx.dataset.DataSetException`, `com.borland.jb.util.ErrorResponse`)

public void updateError(`DataSet` dataSet, `ReadWriteRow` row, `ReadRow` oldRow, `ReadWriteRow` updRow, `DataSetException` ex, `ErrorResponse` response)

This method is called when an exception is thrown during resolution of modifications to a row in the *DataSet*.

dataSet The original *DataSet* passed in to be resolved, positioned at the row that caused the error. This can be used to position any controls bound to it if user interaction is needed.

row The row with the problem. This can be modified to correct the problem and retry the operation.

oldRow The original row.

<i>updRow</i>	The row to use for the next update query if a retry response is chosen. This defaults to the original row (as fetched from the data source before any changes were made). Pass this to the <i>DataSet.refetchRow()</i> method, which refetches updates made to this row by other users since the time the <i>DataSet</i> was originally provided. This is useful in dealing with the situation where another user has modified the same row. It allows your application to do a three-way merge of <i>oldRow</i> , <i>updRow</i> , and <i>row</i> .
<i>ex</i>	The exception that caused the error. Note that this may be a chained exception.
<i>response</i>	Specify a response of <i>ABORT</i> , <i>IGNORE</i> , or <i>RETRY</i> for this error. These values are defined in <i>util.ErrorResponse</i> . Note that an <i>ABORT</i> response causes all insert, update, and delete operations in the same transaction to be rolled back.

See also *com.borland.dx.dataset.DataSetException*

updatingRow(*com.borland.dx.dataset.ReadWriteRow*, *com.borland.dx.dataset.ReadRow*, *com.borland.dx.dataset.ResolverResponse*)

```
public void updatingRow(ReadWriteRow row, ReadRow oldRow, ResolverResponse response)
```

This method is called just before modifications to a row in the data set are resolved on the server.

<i>row</i>	The row that has been modified.
<i>oldRow</i>	The original row.
<i>response</i>	Specify a response of <i>ABORT</i> , <i>IGNORE</i> , or <i>RETRY</i> for this error. These constants are defined in <i>util.ErrorResponse</i> .

ResolverResponse component

dx.dataset package

Extends *com.borland.jb.util.ErrorResponse*

This component is used to collect a response from another component. It encapsulates result information from a resolution event: *RESOLVE*, *SKIP*, or *ABORT*.

ResolverResponse variables

Variable	Defined in
ABORT	com.borland.jb.util.ErrorResponse
IGNORE	com.borland.jb.util.ErrorResponse
response	com.borland.jb.util.ErrorResponse
RETRY	com.borland.jb.util.ErrorResponse

ResolverResponse constructors

ResolverResponse()

public ResolverResponse()

Creates a *ResolverResponse* object.

ResolverResponse properties

Property	Implemented in
abort*	com.borland.jb.util.ErrorResponse
class*	java.lang.Object
ignore*	com.borland.jb.util.ErrorResponse
resolve*	this class
response*	com.borland.jb.util.ErrorResponse
retry*	com.borland.jb.util.ErrorResponse
skip*	this class

resolve

public final boolean isResolve()

Calls the *resolve()* method, indicating that the row should be resolved.

skip

public final boolean isSkip()

Calls the *skip()* method, indicating that the row should be ignored.

ResolverResponse methods

Method	Implemented in
abort()	com.borland.jb.util.ErrorResponse
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
ignore()	com.borland.jb.util.ErrorResponse
notify()	java.lang.Object
notifyAll()	java.lang.Object
resolve()	this class
retry()	com.borland.jb.util.ErrorResponse
skip()	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

resolve()

public final void resolve()

Indicates that a row should be resolved.

skip()

public final void skip()

Indicates that a row should not be resolved.

ResponseAdapter class

dx.dataset package

Extends java.lang.Object

Implements com.borland.dx.dataset.ResponseListener, java.util.EventListener

This is an adapter class for *ResponseListener*, which is used to request a response from the user.

ResponseAdapter properties

Property	Implemented in
class*	java.lang.Object

ResponseAdapter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
response(com.borland.dx.dataset.ResponseEvent)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ResponseEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

The *ResponseEvent* class is used for collecting a response from an application about how to deal with error conditions, table restructuring operations, key violations, etc.

ResponseEvent variables

Variable	Defined in
CANCEL	this class
COMMIT_ON_CLOSE	this class
DATASTORE_ALREADY_OPEN	this class
DATASTORE_CAN_REOPEN	this class
DATASTORE_RECOVERING	this class
DROP_LOG	this class

Variable	Defined in
FILE_EXISTS	this class
IGNORE_ALL	this class
IOEXCEPTION	this class
OK	this class
READ_ONLY_OPEN	this class
source	java.util.EventObject
TYPE_CHANGE_DATA_LOSS	this class
TYPE_CHANGE_PARSE_ERROR	this class
TYPE_CHANGE_PARSE_ERROR_TOTAL	this class
TYPE_CHANGE_PRECISION_LOSS	this class

CANCEL

public static final int CANCEL = 2

Response code requesting that the operation be cancelled.

COMMIT_ON_CLOSE

public static final int COMMIT_ON_CLOSE = 12

DataStoreConnection is closing and the current transaction is about to be committed.

- 1 Call *ok()* to proceed with the *commit()*.
- 2 Call *cancel()* to cause *rollback()* to be called on the connection.

DATASTORE_ALREADY_OPEN

public static final int DATASTORE_ALREADY_OPEN = 5

The *DataStore* appears to already be open by this process or another process. Call *ok()* to attempt to determine if the *DataStore* is really still open. If it is determined that the *DataStore* is not open, a *DATASTORE_CAN_REOPEN ResponseEvent* is sent, and the open process continues. Call *cancel()* to cause this open operation to fail.

DATASTORE_CAN_REOPEN

public static final int DATASTORE_CAN_REOPEN = 6

The *DataStore* was left open, but the process that had it open has terminated. This message comes after *DATASTORE_ALREADY_OPEN*. Call *ok()* to continue the *DataStore* open operation. Call *cancel()* to cause this open operation to fail.

DATASTORE_RECOVERING

```
public static final int DATASTORE_RECOVERING = 11
```

The *DataStore.shutdown()* method was not called for the source *DataStore* by the last process that accessed it. This notification comes just before the system attempts to recover. Call *ok()* to proceed with recovery. Call *cancel()* to cause this operation to fail.

DROP_LOG

```
public static final int DROP_LOG = 10
```

A *DataStore* log file is about to be deleted because it is no longer needed for any active transaction or for crash recovery. This will be called for the “A” log files, “B” log files (if the log is being duplexed), and status log files (if status logging is enabled). Call the *ok()* method if you have deleted the file so that the operation can continue. Call the *cancel()* method to cause this operation to fail.

FILE_EXISTS

```
public static final int FILE_EXISTS = 9
```

Operation cannot continue because the file name in the message already exists. Call *ok()* if you have deleted the file so that the operation can continue. Call *cancel()* to cause this operation to fail.

IGNORE_ALL

```
public static final int IGNORE_ALL = 3
```

Response code requesting that any more error/response requests of this code should be ignored for the duration of this operation.

IOEXCEPTION

```
public static final int IOEXCEPTION = 7
```

The *DataStore* sends this response out whenever there is a read or write failure. A common source for this response event would be when there is insufficient disk space to increase the size of the *DataStore* file when needed. The *source* of the error is set to the *DataStore* instance and exception is set to the *IOException* that was encountered. Call *ok()* to have the I/O operation retried. Call *cancel()* to cause this operation to fail.

OK

```
public static final int OK = 1
```

Response code requesting that the operation be continued.

READ_ONLY_OPEN

```
public static final int READ_ONLY_OPEN = 13
```

The *DataStore* file cannot be opened. An attempt to open the file in read-only mode is about to be made.

- 1 Call *ok()* to proceed open the file in *readOnly* mode.
- 2 Call *cancel()* to fail to open the file and to throw an exception.

TYPE_CHANGE_DATA_LOSS

```
public static final int TYPE_CHANGE_DATA_LOSS = 1
```

The restructure operation is converting data from one type to another. Old values will not be converted to the new data type.

- 1 Call *ok()* to continue.
- 2 Call *cancel()* to abort.
- 3 Call *ignoreAll()* to continue and ignore any future messages of this type.

TYPE_CHANGE_PARSE_ERROR

```
public static final int TYPE_CHANGE_PARSE_ERROR = 3
```

The restructure operation is converting data from one type to another. A parse error occurred while converting a String data type to a non-String data type.

- 1 Call *ok()* to continue.
- 2 Call *cancel()* to abort.
- 3 Call *ignoreAll()* to continue and ignore any future messages of this type.

TYPE_CHANGE_PARSE_ERROR_TOTAL

```
public static final int TYPE_CHANGE_PARSE_ERROR_TOTAL = 4
```

The restructure operation encountered one or more *TYPE_CHANGE_PARSE_ERROR* occurrences. The error is sent out just before the restructure operation is about to be committed.

- 1 Call *ok()* to continue.
- 2 Call *cancel()* to abort the restructure operation.
- 3 Call *ignoreAll()* to continue and ignore any future messages of this type.

TYPE_CHANGE_PRECISION_LOSS

public static final int TYPE_CHANGE_PRECISION_LOSS = 2

The restructure operation is converting data from one type to another. This conversion might result in a loss of precision when values of the old data type are converted to values of the new data type.

- 1 Call *ok()* to continue.
- 2 Call *cancel()* to abort the restructure operation.
- 3 Call *ignoreAll()* to continue and ignore any future messages of this type.

ResponseEvent constructors

ResponseEvent(java.lang.Object, int, java.lang.Exception)

public ResponseEvent(Object source, int code, Exception ex)

Constructs a *ResponseEvent* object.

- source* The object that generates the *ResponseEvent*.
- code* The integer code associated with this event. Pre-defined constants for this property are listed under *ResponseEvent* variables.
- ex* The exception associated with this event.

ResponseEvent(java.lang.Object, int, java.lang.String)

public ResponseEvent(Object source, int code, String message)

Constructs a *ResponseEvent* object.

- source* The object that generates the *ResponseEvent*.
- code* The integer code associated with this event. Pre-defined constants for this property are listed under *ResponseEvent* variables.
- message* The String message associated with this event.

ResponseEvent properties

Property	Implemented in
cancel*	this class
class*	java.lang.Object
code*	this class
exception*	this class
exceptionChain*	com.borland.jb.util.DispatchableEvent

Property	Implemented in
ignoreAll*	this class
message*	this class
ok*	this class
response*	this class
source*	java.util.EventObject

cancel

public final boolean isCancel()

Read-only property that returns **true** if the response is *CANCEL*.

code

public final int getCode()

Read-only property that returns the code value of the *ResponseEvent*. See code constants listed under *ResponseEvent* variables.

exception

public final Exception getException()

Returns an *Exception* if set. Otherwise this method returns null.

ignoreAll

public final boolean isIgnoreAll()

Read-only property that returns **true** if the response is *IGNORE_ALL*.

message

public final String getMessage()

Read-only property that returns String information on what needs to be responded to.

ok

public final boolean isOk()

Read-only property that returns **true** if response is *OK*.

response

public final int getResponse()

Read-only property that returns the response setting of *OK* or *CANCEL*.

ResponseEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
cancel()	this class
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
ignoreAll()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
ok()	this class
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

cancel()

public final void cancel()

Fails the operation. An *Exception* may be thrown to cancel the operation.

dispatch(java.util.EventListener)

public void dispatch(EventListener listener)

This method is an implementation of *DispatchableEvent* that an *EventMulticaster* uses to dispatch an event of this type to the *listener*.

listener The listener to dispatch this event to.

See also *com.borland.jb.util.DispatchableEvent, com.borland.jb.util.EventMulticaster*
Overrides *com.borland.jb.util.DispatchableEvent.dispatch(java.util.EventListener)*

ignoreAll()

public final void ignoreAll()

Causes all future errors/response requests with this event’s code to be ignored. The operation then continues, if possible.

ok()

```
public final void ok()
```

Acknowledges receipt of the *ResponseEvent*. The operation then continues, if possible.

ResponseListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.ResponseAdapter

The *ResponseListener* interface listens to events generated by the *ResponseEvent* class when requesting a response from the user.

ResponseListener methods

Method	Implemented in
response(com.borland.dx.dataset.ResponseEvent)	this class

response(com.borland.dx.dataset.ResponseEvent)

```
public void response(ResponseEvent response)
```

This method is used to request a response from the user. The types of methods are documented in *ResponseEvent*.

RowFilterListener interface

dx.dataset package

Extends java.util.EventListener

This interface is used as a notification when a row is being added or updated. The *RowFilterListener* only controls which rows are displayed in the current view of a *DataSet* based on current filter criteria. The *RowFilterListener* does not delete rows from a *DataSet* or block any data values from being entered. If a newly inserted row contains a value that does not meet the filter criteria, it is stored in the *DataSet*, but does not show in the current view. If you need to prevent rows that do not meet the filter criteria from being stored in a *DataSet*, use the *EditListener.adding()* and *updating()* events.

RowFilterListener methods

Method	Implemented in
<code>filterRow(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.RowFilterResponse)</code>	this class

`filterRow(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.RowFilterResponse)`

`public void filterRow(ReadRow row, RowFilterResponse response)`

This method is called for each row as a data set is opened, and whenever a new or modified row is posted. The *filterRow()* method decides if the current row of the data set should be included in the view. To include it, call *RowFilterResponse.add()*. To exclude it, call *response.ignore()*, which is the default behavior. A *filterRow()* method that never calls *RowFilterResponse.add()* produces an empty *DataSetView*.

- row*
- The current row of the data set.
- response*
- Whether or not the row should be included in the current, filtered, view.

RowFilterResponse class

dx.dataset package

Extends `java.lang.Object`

This class includes or excludes the current row. Rows that are not displayed in the current view are not removed from the *DataSet*, only from the current, filtered view of a *DataSet*. If a newly inserted row contains a value that excludes it from current filter criteria, it is stored in the *DataSet*, but is not displayed in the current view when posted.

This class is usually called from the *DataSet* object's *filterRow* event. You restrict the rows included in a view by adding a *RowFilterListener* and using it to define which rows should be shown. The default action in a *RowFilterListener* is to exclude the row. Your code should call the *RowFilterResponse*'s *add()* method for every row that should be included in the view.

Example The following code sample, taken from “Filtering data” in the *Database Application Developer’s Guide*, demonstrates one use of this class.

```
void queryDataSet1_filterRow(ReadRow readRow, RowFilterResponse rowFilterResponse)
    throws DataSetException
{
    if (formatter == null || columnName == null || columnValue == null ||
        columnName.length() == 0 || columnValue.length() == 0)

        // user-specified filter values are all blank, so all rows are to be
        // included in the DataSetView
        rowFilterResponse.add();
    else {
        readRow.getVariant(columnName, v); // fetches row's value of column
        String s = formatter.format(v);    // formats this to a string

        if (columnValue.equals(s)          // true means show this row
            rowFilterResponse.add();
        else rowFilterResponse.ignore();
    }
}
```

The *filterRow()* method is called for each row as a *DataSet* is opened, and whenever a new or modified row is posted. The *filterRow()* method decides if the current row of the *DataSet* should be included in the view. The *rowFilterResponse.add()* method call adds it. To exclude it, call *rowFilterResponse.ignore()*. Because the *ignore()* method is the default, it is not necessary to explicitly add the **else** clause referencing it. It was added in this example to clarify usage.

RowFilterResponse properties

Property	Implemented in
class*	java.lang.Object

RowFilterResponse methods

Method	Implemented in
add()	this class
canAdd()	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
ignore()	this class
notify()	java.lang.Object

Method	Implemented in
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add()

public final void add()

Call this method inside the *filterRow()* method to cause the row to be included in the current *DataSetView* (i.e. in the filtered view of the *DataSet*).

canAdd()

public final boolean canAdd()

This method returns **true** if the row should be added to the *DataSetView*, otherwise, it returns **false**.

ignore()

public final void ignore()

Call this method inside the *filterRow()* method to cause the row to be excluded in the *DataSetView* (i.e. in the filtered view of the *DataSet*).

RowIterator class

dx.dataset package

Extends com.borland.dx.dataset.ReadWriteRow

Implements java.io.Serializable

RowIterator provides lightweight (low memory usage and fast binding) iteration capabilities for any class that extends *ReadRow*. *RowIterators* can also be used to ensure static type safe access to columns.

RowIterator capabilities are dependent on what class it is bound to. It can provide:

- Column level read operations for any class that extends *ReadRow*.
- Column level read/write operations for any class that extends *ReadWriteRow*.
- Navigation for any class that extends *DataSet*.

A *RowIterator* can be bound (using the *bind* method) to any class that extends *ReadRow*.

Using a *RowIterator* bound to a *DataSet*

If *RowIterator* is bound to a *DataSet*, the *post()* method must be used to cause the changes to appear in the bound *DataSet*. This allows multiple *RowIterators* to be simultaneously editing different rows in the same *DataSet*.

When bound to a *ReadWriteRow*, setting columns takes immediate effect. You can force a *DataSet* to be bound as a *ReadWriteRow* by calling *RowIterator.bind((ReadWriteRow)DataSet)*.

If you have bound a *RowIterator* to a *DataSet*, you must call *RowIterator.unbind()* to free up allocated memory resources used by the *RowIterator*. Note that this does not have to be done if the *DataSet* was bound as a *ReadWriteRow*.

You can extend *RowIterator* or embed it in your own row object. With your own row object, you can add type safe accessor methods with descriptive names. Once established, tools like JBuilder can provide accessor choices inside the editor via Code Insight.

For example, in the case of a customer table, your application may set up an entity *Object* that extends *StorageDataSet*. The entity *Object* contains the business logic for customers expressed as property and event settings with associated business logic. The entity *Object* has a private *jblnit()* method so it is designable using tools like JBuilder. The *jblnit()* method contains the persistent columns, property and event settings for the entity *Object*.

RowIterator properties

Property	Implemented in
<i>assignedNull**</i>	<i>com.borland.dx.dataset.ReadWriteRow</i>
<i>class*</i>	<i>java.lang.Object</i>
<i>columnCount*</i>	<i>com.borland.dx.dataset.ReadRow</i>
<i>columns*</i>	<i>com.borland.dx.dataset.ReadRow</i>
<i>dataSet*</i>	this class
<i>editing*</i>	this class
<i>editingNewRow*</i>	this class
<i>internalRow*</i>	this class
<i>readRow*</i>	this class
<i>readWriteRow*</i>	this class
<i>row*</i>	this class
<i>unassignedNull**</i>	<i>com.borland.dx.dataset.ReadWriteRow</i>

dataSet

```
public final DataSet getDataSet()
```

Read-only property that returns the *DataSet* this *RowIterator* is bound to.

editing

```
public final boolean isEditing()
```

Read-only property that returns whether the *DataSet* is being edited or not.

editingNewRow

```
public final boolean isEditingNewRow()
```

Read-only property that returns whether data is being added to a new row in the *DataSet* or not.

internalRow

```
public final long getInternalRow()
```

Returns a unique identifier for the current row.

Will throw a *DataSetException* if not bound to a *DataSet*.

readRow

```
public final ReadRow getReadRow()
```

Read-only property that returns the *ReadRow* this *RowIterator* is bound to.

readWriteRow

```
public final ReadWriteRow getReadWriteRow()
```

Read-only property that returns the *ReadWriteRow* this *RowIterator* is bound to.

row

```
public final int getRow()
```

Returns the current row position.

Will throw a *DataSetException* if not bound to a *DataSet*.

RowIterator methods

Method	Implemented in
atFirst()	this class
atLast()	this class
bind(com.borland.dx.dataset.DataSet)	this class
bind(com.borland.dx.dataset.ReadRow)	this class
bind(com.borland.dx.dataset.ReadWriteRow)	this class
bind(com.borland.dx.dataset.RowIterator)	this class
cancel()	this class
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteRow()	this class
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	this class
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hashCode()	java.lang.Object
inBounds()	this class
insertRow()	this class
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.ReadRow
isModified(java.lang.String)	com.borland.dx.dataset.ReadRow
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	this class
locate(com.borland.dx.dataset.ReadRow, int)	this class

Method	Implemented in
next()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
post()	this class
prior()	this class
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.ReadWriteRow
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
toString()	com.borland.dx.dataset.ReadRow
unbind()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

atFirst()

public final boolean atFirst()

Returns **true** if this *RowIterator* is at the first row visible by the *DataSet* it is bound to.

atLast()

public final boolean atLast()

Returns **true** if this *RowIterator* is at the last row visible by the *DataSet* it is bound to.

bind(com.borland.dx.dataset.DataSet)

public void bind(DataSet dataSet)

Binds to a *DataSet*.

unbind() must be called when you are done with this iterator. Once the *RowIterator* is bound, set/get operations are allowed on *Columns* and row navigation is allowed.

Edited and inserted rows must call the *post()* method to make the changes visible to the bound *DataSet*.

bind(com.borland.dx.dataset.ReadRow)

public void bind(ReadRow readRow)

Binds to a *ReadRow*. Once the *RowIterator* is bound, get operations are allowed on *Columns*.

bind(com.borland.dx.dataset.ReadWriteRow)

```
public void bind(ReadWriteRow writeRow)
```

Binds to a *ReadWriteRow*. Once the *RowIterator* is bound, set/get operations are allowed on *Columns*.

bind(com.borland.dx.dataset.RowIterator)

```
public void bind(RowIterator iterator)
```

Binds to another *RowIterator*. If the iterator is bound to a *DataSet*, *unbind()* must be called when you are done with this iterator.

cancel()

```
public final void cancel()
```

Cancels edit mode and any edits made to the *RowIterator*.

Will throw a *DataSetException* if not bound to a *DataSet*.

deleteRow()

```
public final void deleteRow()
```

Deletes the row where the *RowIterator* is positioned.

Will throw a *DataSetException* if not bound to a *DataSet*.

first()

```
public final void first()
```

Moves to the first row visible by this *DataSet*.

This may cause *inBounds()* to return **true** if more than one row exists.

inBounds()

```
public final boolean inBounds()
```

Returns **true** if the most recent navigation was in bounds.

insertRow()

```
public final void insertRow()
```

Associates the *RowIterator* with an unposted row containing default values. Columns that have no default values are initialized with *Variant.UNASSIGNED_NULL*.

Will throw a *DataSetException* if not bound to a *DataSet*.

last()

public final void last()

Moves to the last row visible by this *DataSet*.

This may cause *inBounds()* to return **true** if more than one row exists.

locate(com.borland.dx.dataset.ReadRow, int)

public final boolean locate(ReadRow rowLocate, int locateOptions)

Positions the *RowIterator* to the row with values specified by *rowLocate*.

This method behaves much the same as *DataSet.locate()*.

Will throw a *DataSetException* if not bound to a *DataSet*.

next()

public final boolean next()

Moves to the next row visible by this *RowIterator*.

This will cause *inBounds()* to return **false** if *next()* is called when the iterator is positioned at the last visible row.

post()

public final void post()

Terminates edit mode for *RowIterator* and causes new or edited row changes to be posted to the bound *DataSet*.

This method is most useful when bound to a *DataSet*.

prior()

public final boolean prior()

Moves to the prior row visible by this *RowIterator*.

This will cause *inBounds()* to return **false** if *prior()* is called when the iterator is positioned at the first visible row.

unbind()

public final void unbind()

This must be called to free up allocated resources when bound to a *DataSet*. Note that if an iterator is bound to a different *Object* than extends from *ReadRow*, an implicit *unbind()* call will be made for the previously bound object.

RowStatus interface

dx.dataset package

All rows of a *DataSet* have status settings that are used by *DataSet* and other classes. Other status settings track the edit state of row, for example, *INSERTED*, *UPDATED* or *DELETED*. The edit states are used by the *DataSet* class when resolving new and updated data back to its source.

RowStatus settings are used to track edits to rows in a *DataSet*, for example, *INSERTED*, *UPDATED*, or *DELETED* rows will be tracked. *DataSets* can also be filtered by *RowStatus* settings. For more information, see *StorageDataSet.getDeletedRows(...)*, *StorageDataSet.getUpdatedRows(...)*, and *StorageDataSet.getInsertedRows(...)*.

Resolver components like the *QueryResolver* component used by *QueryDataSet* to save changes need *RowStatus* information to know how to save changes back to a JDBC data source. For more information, see *StorageDataSet.startLoading(...)*, *StorageDataSet.endLoading()*, *StorageDataSet.loadRow(...)* to see how rows can be added with a specified row status.

RowStatus variables

Variable	Defined in
DEFAULT	this class
DEFAULT_HIDDEN	this class
DELETED	this class
INSERTED	this class
LOADED	this class
ORIGINAL	this class
PENDING_RESOLVED	this class
UPDATED	this class

DEFAULT

```
static final int DEFAULT = RowStatus.UPDATED|RowStatus.INSERTED|RowStatus.LOADED)
```

This variable is used internally by other *com.borland* classes. You should never use this variable directly.

DEFAULT_HIDDEN

```
static final int DEFAULT_HIDDEN = RowStatus.DELETED|RowStatus.ORIGINAL
```

This variable is used internally by other *com.borland* classes. You should never use this variable directly.

DELETED

public static final int DELETED = 0x01

The row has been deleted.

INSERTED

public static final int INSERTED = 0x04

The row was added after the *DataSet* was loaded.

LOADED

public static final int LOADED = 0x08

The row was loaded (such as from the execution of a QueryDataSet’s JDBC query or an import operation).

ORIGINAL

public static final int ORIGINAL = 0x10

This is the original copy of a changed row.

PENDING_RESOLVED

public static final int PENDING_RESOLVED = 0x200

Row is pending resolution. Used internally.

UPDATED

public static final int UPDATED = 0x02

Row has been changed.

Sort interface

dx.dataset package

The *Sort* interface collects constants used when sorting data.

Sort variables

Variable	Defined in
CASEINSENSITIVE	this class
NOT_NULL	this class

Variable	Defined in
PRIMARY	this class
SORT_AS_INSERTED	this class
UNIQUE	this class

CASEINSENSITIVE

public static final int CASEINSENSITIVE = 0x01

For case-insensitive ordering. Applies to all columns of *String* type.

NOT_NULL

public static final int NOT_NULL = 0x04

Not null constraint on the *sortKey* values.

PRIMARY

public static final int PRIMARY = 0x08|UNIQUE|NOT_NULL

Combines *UNIQUE* and *NOT_NULL* constraints. Only one index per *StorageDataSet* can be created with the *PRIMARY* option enabled.

SORT_AS_INSERTED

public static final int SORT_AS_INSERTED = 0x10

Used as an ordering tie-breaker when there are *sortKeys* with duplicate values or when no *sortKeys* are specified.

UNIQUE

public static final int UNIQUE = 0x02

Unique constraint on the *sortKey* values. A row with column value for *sortKey* that is not unique cannot be added to the *DataSet*.

SortDescriptor class

dx.dataset package

Extends java.lang.Object

Implements java.io.Serializable

The *SortDescriptor* class describes the order by which rows of data that are visible to a *DataSet* are accessed and presented. Sorting data is very easy and fast since indexes are built as they are needed.

The *DataSet* can automatically reposition a new or updated row within the cursor based on the ordering of data by specified columns. In such instances, a row may “fly-away” to its correct position in the *DataSet*.

In an ascending sort, **null** values appear at the bottom of the sort order.

There are no write-accessors for properties of the *SortDescriptor*. To set its properties, use a *SortDescriptor* constructor that takes the appropriate property as a parameter.

SortDescriptor constructors

SortDescriptor(com.borland.dx.dataset.SortDescriptor)

```
public SortDescriptor(SortDescriptor desc)
```

Constructs a *SortDescriptor* that contains the same values as the specified *SortDescriptor*.

desc The *SortDescriptor* to clone properties values from.

SortDescriptor(java.lang.String)

```
public SortDescriptor(String indexName)
```

Constructs a *SortDescriptor* with the specified sort. Defaults to case-sensitive, ascending.

indexName The *String* name of this index.

SortDescriptor(java.lang.String, java.lang.String[], boolean[], boolean, boolean, java.lang.String)

```
public SortDescriptor(String indexName, String[] sortKeys, boolean[] descending, boolean
    caseInsensitive, boolean unique, String localeName)
```

Constructs a *SortDescriptor*, named with the specified *indexName*, with the specified sort keys, as specified in its parameters.

indexName A unique name for this index.

sortKeys An array of columns on which to sort the data.

descending An array of booleans that indicate which *sortKeys* columns are sorted in descending order. The dimension of *sortKeys* and *descending* should be equal if they are both non-null.

caseInsensitive If true, data ordering of *String* type will not be sensitive to the case of the data. If false, the data will be ordered with respect to the case of the *String*.

<i>unique</i>	A constraint on column values. A row with column value for <i>sortKeys</i> that is not unique cannot be added to the <i>DataSet</i> .
<i>localeName</i>	The name (<i>Locale.toString()</i>) of a locale to use for ordering. It is only respected for <i>DataStore</i> . <i>MemoryStore</i> will always use the <i>Locale</i> of the associated <i>StorageDataSet</i> .

SortDescriptor(java.lang.String, java.lang.String[], boolean[], java.lang.String, int)

public SortDescriptor(String indexName, String[] sortKeys, boolean[] descending, String localeName, int options)

Constructs a *SortDescriptor*, named with the specified *indexName*, with the specified sort keys, as specified in its parameters, using the specified options.

<i>indexName</i>	A unique name for this index.
<i>sortKeys</i>	An array of columns on which to sort the data.
<i>descending</i>	An array of booleans that indicate which <i>sortKeys</i> columns are sorted in descending order. The dimension of <i>sortKeys</i> and <i>descending</i> should be equal if they are both non-null.
<i>localeName</i>	The name (<i>Locale.toString()</i>) of a locale to use for ordering. It is only respected for <i>DataStore</i> . <i>MemoryStore</i> will always use the <i>Locale</i> of the associated <i>StorageDataSet</i> .
<i>options</i>	<i>Sort</i> variables. Variables can be combined with the or operator.

SortDescriptor(java.lang.String[])

public SortDescriptor(String[] sortKeys)

Constructs a *SortDescriptor* with the specified sort keys. Defaults to case sensitive, ascending. Case-sensitivity applies for all specified *String* columns. Ascending/descending applies to all specified columns.

<i>sortKeys</i>	The <i>String</i> array containing the names of the <i>Column</i> components by which to sort the data.
-----------------	---

SortDescriptor(java.lang.String[], boolean, boolean)

public SortDescriptor(String[] sortKeys, boolean caseInsensitive, boolean descending)

Constructs a *SortDescriptor* with properties as specified in its parameters.

<i>sortKeys</i>	The <i>String</i> array containing the names of the <i>Column</i> components by which to sort the data.
<i>caseInsensitive</i>	Whether the sort considers (false) or ignores (true) upper and lower case differences. Valid only for <i>String</i> columns. Defaults to false (case sensitive).
<i>descending</i>	Whether the sort is in ascending (false , the default) or descending (true) order.

SortDescriptor(java.lang.String[], boolean, boolean, java.lang.String)

public SortDescriptor(String[] sortKeys, boolean caseInsensitive, boolean descending, String localeName)

Constructs a *SortDescriptor* with properties as specified in its parameters.

<i>sortKeys</i>	The <i>String</i> array containing the names of the <i>Column</i> components by which to sort the data.
<i>caseInsensitive</i>	Whether the sort considers (false) or ignores (true) upper and lower case differences. Valid only for <i>String</i> columns. Defaults to false (case sensitive).
<i>descending</i>	Whether the sort is in ascending (false , the default) or descending (true) order.
<i>localeName</i>	The <i>String</i> name of the locale used for sorting of the data in the <i>DataSet</i> .

SortDescriptor properties

Property	Implemented in
caseInsensitive*	this class
class*	java.lang.Object
descending*	this class
indexName*	this class
keys*	this class
locale*	this class
localeName*	this class
options*	this class
primary*	this class
sortAsInserted*	this class
unique*	this class

caseInsensitive

```
public final boolean isCaseInsensitive()
```

Read-only property that returns whether the sort considers (**false**) or ignores (**true**) upper and lower case differences. Valid only for *String* columns. This property applies to all applicable *String* columns specified in the *keys* property.

descending

```
public final boolean[] getDescending()
```

Read-only property that returns an array that has the descending value for each key. This property applies to all applicable columns specified in the *keys* property. A value of **null** means that all keys are ascending.

indexName

```
public final String getIndexName()
```

Specifies the name of an index that maintains this sorting. *MemoryStore* will ignore this property.

keys

```
public final String[] getKeys()
```

Read-only property that returns the *String* array containing the names of the *Column* components by which the data is sorted.

locale

```
public final Locale getLocale()
```

Returns or specifies the locale that this *SortDescriptor* was created with.

localeName

```
public final String getLocaleName()
```

Returns or specifies the *String* name of locale that this *SortDescriptor* was created with. If no *localeName* was specified, **null** is returned. This property is ignored for *MemoryStore*. *MemoryStore* always uses the locale of the *StorageDataSet*. *DataStore* (which maintains persistent indexes) respects this setting. If this property is **null**, *DataStore* will behave like *MemoryStore* and use the *Locale* of the *StorageDataSet*.

options

```
public final int getOptions()
```

Read-only property that returns the *Sort* options (variables).

primary

```
public final boolean isPrimary()
```

Read-only property that returns **true** if the *PRIMARY* option is enabled.

sortAsInserted

```
public final boolean isSortAsInserted()
```

Read-only property that returns **true** if the *SORT_AS_INSERTED* option is enabled.

unique

```
public final boolean isUnique()
```

This a constraint on column values. A row with a column value for sortKeys that is not unique cannot be added to the *DataSet*.

SortDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(com.borland.dx.dataset.SortDescriptor, java.util.Locale)	this class
equals(com.borland.dx.dataset.SortDescriptor)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
isDescending(int)	this class
keyCount()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

equals(com.borland.dx.dataset.SortDescriptor)

public final boolean equals(SortDescriptor descriptor)

Determines whether the *SortDescriptor* contains the same property values as the specified *SortDescriptor*.

descriptor The *SortDescriptor* to compare property values against.

equals(com.borland.dx.dataset.SortDescriptor, java.util.Locale)

public final boolean equals(SortDescriptor descriptor, Locale locale)

Checks whether this *SortDescriptor* contains the same values as the *descriptor* specified in the parameter of this method. If *locale* is not specified, the default locale is used. This method returns **true** if the *SortDescriptors* are the same, **false** otherwise. The *SortDescriptors* can be equal in two ways:

- When both descriptors have a non-null *indexName* property that are equal. If *indexNames* are equal, no further tests for equality are made.
- If *indexName* does not match, all of the other properties of the *SortDescriptor* are compared for equality.

isDescending(int)

public final boolean isDescending(int i)

Whether or not the values in column *i* are in ascending or descending order. Returns **true** if key *i* is descending.

keyCount()

public final int keyCount()

Returns the number of *Column* components involved in the sort.

toString()

public String toString()

Returns the *String* representation of the values stored in the *SortDescriptor*.

Overrides java.lang.Object.toString()

StatusEvent class

dx.dataset package

Extends com.borland.jb.util.DispatchableEvent

Implements java.io.Serializable

This class is used to inform listeners of specified types of status messages. Typically, these are informative messages, but if the listener (*StatusListener*) wants to take action for a particular status message, the *getCode()* method can be used to determine the type of message being sent.

StatusEvent variables

Variable	Defined in
CHECKING_DATASTORE	this class
CLEAR	this class
DATA_CHANGE	this class
EDIT_CANCELED	this class
EDIT_STARTED	this class
EXCEPTION	this class
LOADING_DATA	this class
LOCATE_MATCH_FOUND	this class
LOCATE_MATCH_NOT_FOUND	this class
LOCATE_NON_STRING	this class
LOCATE_STRING	this class
LOCATE_USE_ENTER	this class
LOCATE_USE_MIXED_CASE	this class
RESTRUCTURING	this class
SORTING	this class
source	java.util.EventObject

CHECKING_DATASTORE

public static final int CHECKING_DATASTORE = 15

Checking to see if the *DataStore* was not closed properly. This check can take 7–10 seconds.

CLEAR

public static final int CLEAR = 10

Sent by the *DataSet.clearStatus()* method to clear status information, for example, the *StatusBar* control.

DATA_CHANGE

public static final int DATA_CHANGE = 8

Status message when data in a *DataSet* has changed. Use enter key to start the locate operation. Sent by *DataSet.interactiveLocate()*.

EDIT_CANCELED

public static final int EDIT_CANCELED = 12

Cancelled edit for new or existing row of a *DataSet*.

EDIT_STARTED

public static final int EDIT_STARTED = 11

Entered edit state for new or existing row of a *DataSet*.

EXCEPTION

public static final int EXCEPTION = 9

An exception was thrown.

LOADING_DATA

public static final int LOADING_DATA = 1

Status notification about loading rows in a *DataSet*.

LOCATE_MATCH_FOUND

public static final int LOCATE_MATCH_FOUND = 3

Status notification that a match was found during a locate operation. Sent by *DataSet.interactiveLocate()*.

LOCATE_MATCH_NOT_FOUND

public static final int LOCATE_MATCH_NOT_FOUND = 4

Status notification that a match could not be found for a location operation. Sent by *DataSet.interactiveLocate()*.

LOCATE_NON_STRING

```
public static final int LOCATE_NON_STRING = 7
```

Status notification that a locate operation on a non-String column expects a value whose length is greater than zero. Enter a value and press *Enter* to begin search. Sent by *DataSet.interactiveLocate()*.

LOCATE_STRING

```
public static final int LOCATE_STRING = 6
```

Status notification that a locate operation of *Locate.NEXT* or *Locate.PRIOR* on a String column expects a value of length greater than zero. Enter a value and press *Enter* to begin search. Sent by *DataSet.interactiveLocate()*.

LOCATE_USE_ENTER

```
public static final int LOCATE_USE_ENTER = 2
```

Status notification that the user should press *Enter* to perform the locate. This notification is sent by *DataSet.interactiveLocate()* and applies to all locates on non-String columns.

LOCATE_USE_MIXED_CASE

```
public static final int LOCATE_USE_MIXED_CASE = 5
```

Enter a value and use mixed case characters for a case-sensitive search. Sent by *DataSet.interactiveLocate()*.

RESTRUCTURING

```
public static final int RESTRUCTURING = 14
```

Status notification sent by long-running restructure operations.

SORTING

```
public static final int SORTING = 13
```

Status notification sent by long-running sort operations.

StatusEvent constructors

StatusEvent(java.lang.Object, int, java.lang.String)

public StatusEvent(Object source, int code, String message)

A constructor for a *StatusEvent* object with the following parameters:

<i>source</i>	The object generating the notification.
<i>code</i>	The type of message being sent. The int value represents one of the <i>StatusEvent</i> variables.
<i>message</i>	The String text associated with the <i>code</i> .

StatusEvent(java.lang.Object, java.lang.Throwable)

public StatusEvent(Object source, Throwable ex)

A constructor for a *StatusEvent* object with the following parameters:

<i>source</i>	The object generating the notification.
<i>ex</i>	The <i>Exception</i> that occurred.

StatusEvent properties

Property	Implemented in
class*	java.lang.Object
code*	this class
exception*	this class
exceptionChain*	com.borland.jb.util.DispatchableEvent
message	this class
source*	java.util.EventObject

code

public final int getCode()

Gets the code that indicates the type of message. The int value returned represents one of the *StatusEvent* variables.

exception

public final Throwable getException()

Get the exception which caused the message. Null unless code = EXCEPTION *

message

```
public final String getMessage()  
public final void setMessage(String message)
```

Determines the String text associated with the status notification.

StatusEvent methods

Method	Implemented in
appendException(java.lang.Exception)	com.borland.jb.util.DispatchableEvent
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	com.borland.jb.util.DispatchableEvent
toString()	com.borland.jb.util.DispatchableEvent
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

dispatch(java.util.EventListener)

```
public void dispatch(EventListener listener)
```

Sends the *StatusEvent* notification to all registered *StatusListeners*.

Overrides com.borland.jb.util.DispatchableEvent.dispatch(EventListener)

StatusListener interface

dx.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.dataset.DataSetView,
com.borland.dx.dataset.StorageDataSet,
com.borland.dx.dataset.TableDataSet,
com.borland.dx.sql.dataset.ProcedureDataSet,
com.borland.dx.sql.dataset.QueryDataSet

This interface is used as a mechanism by which a status bar gets its information from a *DataSet*. Use this interface to customize or suppress the message sent from the *DataSet* to the status bar.

To use the *StatusListener*,

- 1 Place a *JdbStatusLabel* component in your application.
- 2 Set the *dataSet* property of the *StatusBar* to the *DataSet* whose status you want to display.
- 3 Code a *StatusListener* to customize the messages being sent to the *JdbStatusLabel*.

The *StatusListener* can be used in the following ways:

- To show how many rows have been loaded from the *DataSet*.
- To display search results when a *JdbNavComboBox* or *JdbNavField* is used to find specific data.

StatusListener methods

Method	Implemented in
<code>statusMessage(com.borland.dx.dataset.StatusEvent)</code>	this class

statusMessage(com.borland.dx.dataset.StatusEvent)

`public void statusMessage(StatusEvent event)`

Called when a message is being sent to the status bar and other status listeners.

event The event that prompted the message to be sent.

StorageDataSet component

dx.dataset package

Extends	com.borland.dx.dataset.DataSet
Extended by	com.borland.dx.dataset.TableDataSet, com.borland.dx.sql.dataset.ProcedureDataSet, com.borland.dx.sql.dataset.QueryDataSet
Implements	com.borland.dx.dataset.AccessListener, com.borland.dx.dataset.ColumnDesigner, com.borland.dx.dataset.Designable, com.borland.dx.dataset.MasterNavigateListener, com.borland.dx.dataset.StatusListener, java.io.Serializable, java.util.EventListener

This is an abstract class of core functionality for *StorageDataSet* functionality. It cannot be used directly. See *TableDataSet*, *QueryDataSet*, *ProcedureDataSet* components instead.

StorageDataSet manages a 2-dimensional array of data. *Column* data types are specified through *Column* components. A *StorageDataSet* has a *Column* component for each data column that it contains. *Column* components contain type information for the data columns in a *StorageDataSet*.

The *StorageDataSet* component extends the basic cursor functionality provided by its superclass *DataSet* with the:

- storage of the data (using *MemoryStore* or *DataStore*).
- facility for structural (*Column*) changes to the *StorageDataSet*.
- *provider* and *resolver* properties.
- ability to be the base storage for multiple *DataSetView* components.

The *StorageDataSet* component is extended by *QueryDataSet*, *ProcedureDataSet*, and *TableDataSet* components. The *QueryDataSet* and *ProcedureDataSet* work with a *Database* component to obtain data from a remote server through the execution of a query or stored procedure. You can also load data stored in a text file into a *TableDataSet* object. Once data is loaded (provided) into a *StorageDataSet* object, you handle the data in a common way, regardless of how the data was obtained or which *StorageDataSet* extension you use.

When making structure changes to the *StorageDataSet*, such as setting the *store* property to use a *DataStore*, use the *addColumn*, *changeColumn*, *dropColumn*, and *moveColumn* methods as these methods allow for better before and after mapping. Otherwise, when the *StorageDataSet* is restructured, *Column* names are used for mapping, which may not always work as anticipated. The *Column* component's *preferredOrdinal* property is retained when calling these *StorageDataSet* methods.

If your application involves a master-detail relationship, the *resolveOrder* property indicates the order in which changes made to the *DataSets* are resolved back to the data source.

StorageDataSet constructors

StorageDataSet()

```
public StorageDataSet()
```

Constructs a *StorageDataSet* component, generates a *StatusEvent* of *StatusEvent.LOADING_DATA* and sets the following properties:

- *maxRows* to -1
- *maxDesignRows* to 50
- *metaDataUpdate* to *MetaDataUpdate.ALL*

StorageDataSet properties

Property	Implemented in
allRowIds**	this class
assignedNull**	com.borland.dx.dataset.ReadWriteRow
calcAggFieldsListener*	this class
calcFieldsListener*	this class
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns**	this class
dataFile	this class
defaultValues**	com.borland.dx.dataset.DataSet
deletedRowCount*	this class
detailDataSetWithFetchAsNeeded*	com.borland.dx.dataset.DataSet
details*	com.borland.dx.dataset.DataSet
displayErrors	com.borland.dx.dataset.DataSet
duplicates*	this class
editable	com.borland.dx.dataset.DataSet
editing*	com.borland.dx.dataset.DataSet
editingNewRow*	com.borland.dx.dataset.DataSet
empty*	com.borland.dx.dataset.DataSet
enableDelete	com.borland.dx.dataset.DataSet
enableInsert	com.borland.dx.dataset.DataSet
enableUpdate	com.borland.dx.dataset.DataSet
insertedRowCount*	this class
internalRow*	com.borland.dx.dataset.DataSet
lastColumnVisited	com.borland.dx.dataset.DataSet
locale	this class
masterLink	com.borland.dx.dataset.DataSet
maxDesignRows	this class
maxResolveErrors	this class
maxRows	this class
metaDataUpdate	this class
needsRestructure*	this class
open*	com.borland.dx.dataset.DataSet
provider	this class
readOnly	this class
resolvable	this class
resolveOrder	this class
resolver	this class
row*	com.borland.dx.dataset.DataSet

Property	Implemented in
rowCount*	com.borland.dx.dataset.DataSet
rowFilterListener*	com.borland.dx.dataset.DataSet
schemaName	this class
sort	com.borland.dx.dataset.DataSet
status*	com.borland.dx.dataset.DataSet
storageDataSet*	com.borland.dx.dataset.DataSet
store	this class
storeClassFactory	this class
storeName	this class
tableName	this class
unassignedNull**	com.borland.dx.dataset.ReadWriteRow
updatedRowCount*	this class

allRowIds

public final synchronized void setAllRowIds(boolean setting)

Sets all *Column* components as being unique row identifiers.

calcAggFieldsListener

public final CalcAggFieldsListener getCalcAggFieldsListener()

Read-only property that returns the *CalcAggFieldsListener* of the *StorageDataSet*.

calcFieldsListener

public final CalcFieldsListener getCalcFieldsListener()

Read-only property that returns the *CalcFieldsListener* of the *StorageDataSet*.

columns

public synchronized void setColumns(Column[] columns)

Specifies the *Column* components in the *StorageDataSet* and sets the *persist* property to **true** for all specified columns. Any pre-existing columns are removed when this property is set.

dataFile

public final DataFile getDataFile()
public void setDataFile(DataFile dataFile)

Specifies the *DataFile* implementation used for file import and export operations. See *TextDataFile* for an implementation of *DataFile*.

deletedRowCount

```
public final int getDeletedRowCount()
```

Read-only property that returns the number of deleted rows not visible to this *StorageDataSet*.

duplicates

```
public final StorageDataSet getDuplicates()
```

When a unique sort property is applied for the first time, rows that violate the unique constraint will be copied off to a separate duplicates *DataSet*. The duplicates *DataSet* can be retrieved by calling `getDuplicates()`.

insertedRowCount

```
public final int getInsertedRowCount()
```

Read-only property that returns the number of inserted rows visible to this *StorageDataSet*.

locale

```
public Locale getLocale()
public synchronized void setLocale(Locale locale)
```

Used for formatting the data stored in the *StorageDataSet*. Locale contains country or area-specific formatting specifications such as date format (MM/DD/YY, DD/MM/YY, YY/MM/DD), currency symbol, and so on.

When set at the *StorageDataSet* level, this property is the default *locale* for all *Column* components in the *StorageDataSet*. If the *StorageDataSet*'s *locale* property is **null**, and the *store* property is set to a *DataStore* component, the *locale* property of the *DataStore* will be used if set. If not set any level, it defaults to the locale of the Java environment.

maxDesignRows

```
public final int getMaxDesignRows()
public final void setMaxDesignRows(int maxDesignRows)
```

Limits the number of rows that can be initially loaded into a *DataSet* in JBuilder's UI designer. The default is to load 50 rows. If the limit is reached, only that number of rows is displayed; no message is displayed nor is any *Exception* thrown.

maxResolveErrors

```
public final int getMaxResolveErrors()
public final void setMaxResolveErrors(int maxResolveErrors)
```

The maximum number of errors that can be logged in an error *DataSet* before the *ResolutionManager* is called.

If this property is set to 0, no errors will be logged. If set to -1, all errors will be logged.

See also *ResolveError*

maxRows

```
public final int getMaxRows()
public final void setMaxRows(int maxRows)
```

Limits the number of rows that can initially be loaded into the *DataSet* when running the application or applet. Also called a governor. The *setMaxRows()* method has no effect on rows added after the initial loading of data, for example, from a query or import specification.

This property defaults to -1 which indicates that there is no limit to the number of rows that can be initially loaded into a *DataSet*. No message is displayed nor is any *Exception* thrown when the limit is reached, however, only that number of rows is included in the *DataSet* when it is loaded with data. With long running queries that can possibly return large result sets, it is advisable to set the *maxRows* property since the maximum number of rows defaults to unlimited.

See also *Load.AS_NEEDED*

metaDataUpdate

```
public final int getMetaDataUpdate()
public final synchronized void setMetaDataUpdate(int metaDataUpdate)
```

Determines what kind of metadata discovery is performed when executing a query or a stored procedure against a SQL server database. Valid values for this property are defined in the *MetaDataUpdate* interface.

To prevent the addition of row ID columns and various metadata related properties on *DataSet* and *Column* components, set this property to *MetaDataUpdate.NONE*.

needsRestructure

```
public final boolean getNeedsRestructure()
```

Returns **true** for *StorageDataSet* components when structural changes have occurred (add, drop, move, change column) and the *store* being used requires a restructure. To cause the restructure to happen, call the *restructure()* method. After a successful restructure, *getNeedsRestructure()* returns **false**.

A *StorageDataSet* that uses a *MemoryStore* always returns **false**. Currently, this method is meaningful only with *DataStore* use.

A *StorageDataSet* with pending structural changes can still be used. Moved columns can be read and written to. Deleted columns are not visible, inserted columns can be read, but not written, changed data type columns can be read but not written to.

provider

```
public Provider getProvider()
public void setProvider(Provider provider)
```

Specifies the *Provider* component that controls how the data is fetched from the database when the *StorageDataSet* is opened. JDataStore includes these provider components:

- *QueryProvider*
- *ProcedureProvider*
- *OracleProcedureProvider*

readOnly

```
public final boolean isReadOnly()
public final void setReadOnly(boolean readOnly)
```

Specifies whether edits to the *DataSet* are permitted. If **true**, edits to the *DataSet* are not allowed, otherwise the *DataSet* can be updated.

A *StorageDataSet* can be internally marked *readOnly* in some situations. These automatic *readOnly* settings cannot be cleared by calling *setReadOnly(false)*. The return value of *DataSetException.getMessage()* explains why a *StorageDataSet* is *readOnly*.

A common reason for a *StorageDataSet* to be automatically marked *readOnly* is that a *Provider*, such as *QueryProvider* or *ProcedureProvider*, could not determine which *Columns* should have their *rowId* property set to **true**. This *readOnly* state can be cleared by calling *Column.setRowId()*, *StorageDataSet.setRowId()*, or *StorageDataSet.setAllRowIds()*.

If a *QueryProvider* or *ProcedureProvider* cannot determine the *tableName* property setting, the changes in the *StorageDataSet* cannot be saved back. To save changes back, the *StorageDataSet.setTableName* property must be explicitly set. If the query was the result of a join operation, and you would

like to resolve the changes back, the *Column.tableName* property will have to be set for each column. If this was not successfully accomplished by the *Provider*, this property will have to be set for each *Column* that is to have its values saved (resolved) back.

To prevent edits to the *DataSet* through data-aware controls only (permitting programmatic edits), set the *editable* property.

See also *editable* property of the *DataSet* class, *resolvable* property of this class

resolvable

```
public final boolean isResolvable()
public final void setResolvable(boolean resolvable)
```

Specifies whether to allow changes made to this *DataSet* to be saved back to its data source (**true**) or not (**false**).

resolvable is a user property that indicates an intent to resolve edits to the *StorageDataSet*. If set to false, inserts, updates, and deletes are not tracked. This will save space and improve performance for *StorageDataSets* that are wired to a *DataStore* when you have no intention of resolving.

See also *editable* property of the *DataSet* class, *readOnly* property of this class

resolveOrder

```
public final String[] getResolveOrder()
public final void setResolveOrder(String[] resolveOrder)
```

A *String* array that specifies the resolution order for multi-table resolution. Inserts and updates use the order of the *String* array while deletes use the reverse order. The default order of the tables in the *String* array is the order by which they appear in the query statement. The table names should include any necessary schema names. If a table is removed from the list, then the columns from that table will not be resolved. If your application involves a master-detail relationship, the *resolveOrder* property indicates the order in which changes made to the *DataSets* are resolved back to the data source.

resolver

```
public Resolver getResolver()
public void setResolver(Resolver resolver)
```

Specifies the *Resolver* object to use that defines resolver logic when saving data changes back to the data source.

schemaName

```
public final String getSchemaName()
public synchronized void setSchemaName(String schemaName)
```

Write-only property for the table name of the data source for the *Column* components of this *StorageDataSet*. If a *DataSet* has any updateable *Column* components, this property must reflect the table name of the data source.

store

```
public final Store getStore()
public final void setStore(Store store)
```

Sets the storage for the *DataSet*. Valid values for this property are *MemoryStore* or *DataStore*. The default is *MemoryStore*.

With *MemoryStore*, *DataSet* data is stored in memory and released when an application closes. With *DataStore*, storage is persistent and file-based so that when an application closes, it can be restarted in the same state as when it was closed.

storeClassFactory

```
public final StoreClassFactory getStoreClassFactory()
public final void setStoreClassFactory(StoreClassFactory factory)
```

Allows a *DataStore* table to be accessed only by a specific *StorageDataSet* class or *StorageDataSet* class extension. This allows an application to make sure that property and event settings are always activated when a *DataStore* table is accessed.

Currently, this property is only meaningful when the *StorageDataSet.store* property is set to a *DataStoreConnection* or *DataStore* component.

See also *StoreClassFactory*

storeName

```
public final String getStoreName()
public synchronized void setStoreName(String name)
```

If the *store* property component supports naming of a *DataSet* components in the *Store*, this can be set. The default *store* is *MemoryStore*, which does not support naming of *DataSet* components. This property is meaningful to and used only by *DataStore*.

When specified for a *DataStore* component, the *storeName* property must be unique. The value specified for *storeName* is case insensitive, though it is stored internally in upper case. To force a *storeName* to be case insensitive, enclose it in quotes, for example:

```
" \"name\" "
```

tableName

```
public final String getTableName()
public synchronized void setTableName(String tableName)
```

Stores the name of the table from which the *DataSet* obtains its data. If a *DataSet* has any updateable columns, the name of the table for these columns is stored as *tableName*.

The table name is normally retrieved automatically from the data source. Some stored procedures and queries however will not return the table name. If it doesn't, use *setTableName()* to specify the table name so that changes made to the *DataSet* can be resolved back to the data source, then set the *rowId* and *metadataupdate* properties (if not already set). The table name is also useful as a *String* identifier of the data source.

updatedRowCount

```
public final int getUpdatedRowCount()
```

Read-only property that returns the number of updated rows visible to this *StorageDataSet*.

StorageDataSet methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	com.borland.dx.dataset.DataSet
addColumn(com.borland.dx.dataset.Column)	this class
addColumn(java.lang.String, int)	this class
addColumn(java.lang.String, java.lang.String, int)	this class
addLoadRowListener(listener)	this class
addRow(com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
addUniqueColumn(com.borland.dx.dataset.Column)	this class
allocateValues()	com.borland.dx.dataset.DataSet
atFirst()	com.borland.dx.dataset.DataSet
atLast()	com.borland.dx.dataset.DataSet
cancel()	com.borland.dx.dataset.DataSet
cancelLoading()	this class
cancelOperation()	this class
canNavigate (com.borland.dx.dataset.Column, int)	com.borland.dx.dataset.DataSet
canSet(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
changeColumn(int, com.borland.dx.dataset.Column)	this class

Method	Implemented in
changesPending()	this class
clearStatus()	com.borland.dx.dataset.DataSet
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneColumns()	this class
cloneDataSetStructure()	this class
cloneDataSetView()	com.borland.dx.dataset.DataSet
close()	com.borland.dx.dataset.DataSet
closeProvider(boolean)	this class
columnIsVisible(java.lang.String)	com.borland.dx.dataset.DataSet
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	com.borland.dx.dataset.DataSet
deleteDuplicates()	this class
deleteRow()	com.borland.dx.dataset.DataSet
dittoRow(boolean, boolean)	com.borland.dx.dataset.DataSet
dittoRow(boolean)	com.borland.dx.dataset.DataSet
dropAllIndexes()	this class
dropColumn (com.borland.dx.dataset.Column)	this class
dropColumn(java.lang.String)	this class
dropIndex()	com.borland.dx.dataset.DataSet
dropIndex (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	this class
editRow()	com.borland.dx.dataset.DataSet
empty()	this class
emptyAllRows()	com.borland.dx.dataset.DataSet
emptyRow()	com.borland.dx.dataset.DataSet
enableDataSetEvents(boolean)	com.borland.dx.dataset.DataSet
endLoading()	this class
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	com.borland.dx.dataset.DataSet

Method	Implemented in
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDataRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDataRow(int, com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDeletedRows (com.borland.dx.dataset.DataSetView)	this class
getDetail(java.lang.String)	com.borland.dx.dataset.DataSet
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInsertedRows (com.borland.dx.dataset.DataSetView)	this class
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getOriginalRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	this class

Method	Implemented in
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getUpdatedRows (com.borland.dx.dataset.DataSetView)	this class
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
goToClosestRow(int)	com.borland.dx.dataset.DataSet
goToInternalRow(long)	com.borland.dx.dataset.DataSet
goToRow(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
goToRow(int)	com.borland.dx.dataset.DataSet
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	com.borland.dx.dataset.DataSet
hashCode()	java.lang.Object
hasRowIds()	this class
hasValidations()	com.borland.dx.dataset.DataSet
inBounds()	com.borland.dx.dataset.DataSet
indexExists (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	this class
insertRow(boolean)	com.borland.dx.dataset.DataSet
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	com.borland.dx.dataset.DataSet
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.DataSet
isModified(java.lang.String)	com.borland.dx.dataset.DataSet
isNew(int)	com.borland.dx.dataset.DataSet
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow

Method	Implemented in
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	com.borland.dx.dataset.DataSet
loadRow()	this class
loadRow(int)	this class
locate(com.borland.dx.dataset.ReadRow, int)	com.borland.dx.dataset.DataSet
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	com.borland.dx.dataset.DataSet
masterNavigated(com.borland.dx.dataset.MasterNavigateEvent)	com.borland.dx.dataset.DataSet
masterNavigating(com.borland.dx.dataset.MasterNavigateEvent)	com.borland.dx.dataset.DataSet
moveColumn(int, int)	this class
moveRow(int)	com.borland.dx.dataset.DataSet
next()	com.borland.dx.dataset.DataSet
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.dx.dataset.DataSet
openDetails()	com.borland.dx.dataset.DataSet
post()	com.borland.dx.dataset.DataSet
postAllDataSets()	this class
prior()	com.borland.dx.dataset.DataSet
provideMoreData()	this class
recalc()	this class
refetchRow (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.DataSet
refilter()	com.borland.dx.dataset.DataSet
refresh()	this class
refreshSupported()	this class
removeLoadRowListener(listener)	this class
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
reset()	this class
resetInBounds()	com.borland.dx.dataset.DataSet
resetPendingStatus(boolean)	this class
resetPendingStatus(long, boolean)	this class
restructure()	this class
saveChanges()	com.borland.dx.dataset.DataSet
saveChanges (com.borland.dx.dataset.DataSet)	this class
saveChangesSupported()	this class
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.DataSet
setDisplayVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setRowId(java.lang.String, boolean)	this class
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startEditCheck (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean, boolean)	this class
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean)	this class
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean)	this class
statusMessage (com.borland.dx.dataset.StatusEvent)	com.borland.dx.dataset.DataSet
statusMessage(int, java.lang.String)	com.borland.dx.dataset.DataSet
toggleViewOrder(java.lang.String)	com.borland.dx.dataset.DataSet
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
validate()	com.borland.dx.dataset.DataSet
validate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

addColumn(java.lang.String, int)

```
public final int addColumn(String columnName, int datatype)
```

Adds a *Column* to a *DataSet* where *columnName* indicates the *String* name of the *Column* and *dataType* is the data type of the *Column*. The *Column* is added at the end of all existing columns. The return value *int* indicates the ordinal position of the newly-added *Column*. Use the *enableDataSetEvents* method to propagate changes to a control that is bound to this *DataSet*.

To achieve similar results as when calling *StorageDataSet.setColumns(...)* method, call the *addColumn(...)* method followed by *Column.setPersist(true)*.

If the *store* property is set, *StorageDataSet.restructure()* must be called before this column can be edited.

On error, this method throws a *DataSetException*.

<i>columnName</i>	The <i>String</i> name of the <i>Column</i> component to add to the <i>StorageDataSet</i> .
<i>datatype</i>	The data type of the data in the <i>Column</i> . Valid values for this parameter are defined in <i>com.borland.dx.dataset.Variant</i> variables.

addColumn(java.lang.String, java.lang.String, int)

```
public final int addColumn(String columnName, String caption, int dataType)
```

Adds a *Column* to the end of existing columns in a *DataSet*. Row values for the added *Column* are set to **null**. The return value *int* indicates the ordinal position of the newly-added *Column*. On error, this method throws a *DataSetException*.

If the *store* property is set, *StorageDataSet.restructure()* must be called before this column can be edited.

<i>columnName</i>	The <i>String</i> name of the <i>Column</i> by which you refer to it most of the time.
<i>colLabel</i>	The name of the (displayed) label for the <i>Column</i> in a data-aware control. You may want to use a more descriptive name than <i>columnName</i> .
<i>dataType</i>	The data type of the field to be added. Valid values for this parameter are described in <i>com.borland.dx.dataset.Variant</i> variables.

addLoadRowListener(listener)

```
public void addLoadRowListener(LoadRowListener listener)
```

Adds the specified *LoadRowListener* object.

addUniqueColumn(com.borland.dx.dataset.Column)

```
public final int addUniqueColumn(Column column)
```

Adds a *Column* to the *DataSet* only if it does not already exist. The specified *column* is cloned before being added to the *DataSet*. The return value *int* indicates the ordinal position of the newly-added *Column*. On error, this method throws a *DataSetException* of *COLUMN_TYPE_CONFLICT*.

<i>column</i>	The <i>Column</i> component to add to this <i>StorageDataSet</i> .
---------------	--

cancelLoading()

```
public final void cancelLoading()
```

Cancels any long running load operations currently active on this *DataSet*. For this have an effect, the long running loading operation must be executing on a different thread than the thread that calls this method.

Overrides `com.borland.dx.dataset.DataSet.cancelLoading()`

cancelOperation()

```
public final void cancelOperation()
```

Cancels any long running operation currently active on this *DataSet*. This includes loading a query result, restructure operations, and index building operations. For this have an effect, the long running operation must be executing on a different thread than the thread that calls this method.

Overrides `com.borland.dx.dataset.DataSet.cancelOperation()`

changesPending()

```
public boolean changesPending()
```

Returns **true** if there are any inserted, deleted, or updated rows to be resolved back to the data source. If **true**, one or more of the following methods *getUpdatedRowCount()*, *getDeletedCount()*, or *getInsertedCount()* return a value greater than 0.

cloneColumns()

```
public Column[] cloneColumns()
```

Creates a copy of all *Columns* in the *StorageDataSet*.

cloneDataSetStructure()

```
public final synchronized StorageDataSet cloneDataSetStructure()
```

Creates a new empty *StorageDataSet* object with the identical structure as the current *StorageDataSet*.

closeProvider(boolean)

```
public void closeProvider(boolean loadRemainingRows)
```

This method can be used to terminate a provider that is currently still loading rows into the *StorageDataSet*. If a load option such as *Load.AS_NEEDED* or *Load.ASYNCHRONOUS* is being used by a *QueryProvider* or *ProcedureProvider*, and *closeProvider()* is called with *loadRemainingRows* set to **true**, the remaining rows will be retrieved before the provider is closed. If *loadRemainingRows* is **false**, the provide operation will be terminated and the remaining rows will not be retrieved.

deleteDuplicates()

```
public final void deleteDuplicates()
```

When a unique sort property is applied for the first time, rows that violate the unique constraint will be copied off to a separate duplicates *DataSet*. The duplicates *DataSet* can be retrieved by calling *getDuplicates()*. Opening a *DataSet* with a new *sort* property setting that has a unique constraint will not succeed if a duplicates *StorageDataSet* already exists. This restriction protects an application from accidentally losing valuable data in duplicate rows.

Call *deleteDuplicates()* to delete the old set of duplicates.

dropAllIndexes()

```
public final void dropAllIndexes()
```

Drops all indexes used by this *StorageDataSet*. For *MemoryStore*, this can be used to save memory after viewing the data in different sort orders. For *DataStore*, this frees up maintained secondary indexes, releasing disk storage. In both *MemoryStore* and *DataStore*, this can improve update performance.

dropColumn(com.borland.dx.dataset.Column)

```
public final synchronized void dropColumn(Column column)
```

Deletes the specified *Column* object (and all the data it contains) from the *DataSet*. If the *Column* cannot be dropped, this method throws a *DataSetException*. *dropColumn* does not affect the relative order of columns but does cause a recalculation of column ordinal values.

If the *store* property is set, *StorageDataSet.restructure()* must be called before this column can be edited.

column The *Column* component to drop from this *StorageDataSet*.

dropIndex(com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)

```
public final boolean dropIndex(SortDescriptor descriptor, RowFilterListener listener)
```

This method can be used to drop unneeded indexes used to maintain a sorted and filtered view of a *DataSet*. Dropping an index can save space and improve the performance of insert/add/delete operations.

descriptor Sort settings that describe the index.

listener *RowFilterListener* used for filtering. Pass null if not a filtered index.

empty()

```
public final void empty()
```

Empties all rows of the *DataSet* and resets the *DataSet* to contain no rows. All change state information (inserted, deleted, changed) is lost and therefore, nothing remains in the *DataSet* to be resolved back to the original data source. On error, this method generates a *DataSetException*.

endLoading()

```
public final synchronized void endLoading()
```

Stops the loading of data into the *StorageDataSet*.

See also *startLoading(com.borland.dx.dataset.LoadCancel, int, boolean), loadRow()*

getDeletedRows(com.borland.dx.dataset.DataSetView)

```
public final synchronized void getDeletedRows(DataSetView deleteDataSet)
```

Initializes *deleteDataSet* to display only the deleted rows in the current *DataSet*. On error, this method generates a *DataSetException*.

deleteDataSet The *DataSetView* initialized by this method to include only deleted rows.

getInsertedRows(com.borland.dx.dataset.DataSetView)

```
public final synchronized void getInsertedRows(DataSetView insertDataSet)
```

Initializes the *DataSetView* to display only the inserted (new) rows in the current *DataSet*. On error, this method generates a *DataSetException*.

insertDataSet The *DataSetView* initialized by this method to include only inserted rows.

getOriginalRow(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)

```
public final void getOriginalRow(DataSet updateDataSet, ReadWriteRow originalRow)
```

Given a *DataSet* with a changed record, this method initializes a *dataRow* with the original values.

getUpdatedRows(com.borland.dx.dataset.DataSetView)

```
public final synchronized void getUpdatedRows(DataSetView updateDataSet)
```

Initializes a *DataSetView* object to display only the updated rows in the current *DataSet*. On error, this method generates a *DataSetException*.

updateDataSet The *DataSetView* that is initialized by this method to contain only updated rows.

hasRowIds()

public final synchronized boolean hasRowIds()

Returns whether the *DataSet* has any unique row identifiers. If at least one exists, this method returns **true**.

indexExists(com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)

public final boolean indexExists(SortDescriptor descriptor, RowFilterListener listener)

Returns **true** if a maintained index already exists for the given descriptor and listener.

descriptor

Sort settings that describe the index.

listener

RowFilterListener used for filtering. Pass null if not a filtered index.

loadRow()

public final void loadRow()

This method allows for faster loading of a *StorageDataSet*. It calls *loadRow(loadStatus)*, where *loadStatus* is the value of *loadStatus* passed to *startLoading()*.

See also *startLoading(com.borland.dx.dataset.LoadCancel, int, boolean), endLoading()*

loadRow(int)

public final long loadRow(int status)

Allows for faster loading of a *StorageDataSet*. You can also use this method to load rows as updated or deleted.

To load a row as deleted, set the variant values returned from *startLoading()* and call *loadRow(RowStatus.DELETED)*.

To load a row as updated, set the variant values returned from *startLoading()* to the original row values and call *loadRow(RowStatus.ORIGINAL)*. Then set the variant values returned from *startLoading* to the updated row values and call *loadRow(RowStatus.UPDATED)*.

status

Status of the row, typically *RowStatus.INSERTED* or *RowStatus.LOADED*.

moveColumn(int, int)

```
public final synchronized void moveColumn(int oldOrdinal, int newOrdinal)
```

Moves a *Column* at the specified ordinal position to a new ordinal position. If either parameter is invalid, a *DataSetException* of type *INVALID_COLUMN_POSITION* is thrown.

If the *store* property is set, *StorageDataSet.restructure()* must be called before this column can be edited.

oldOrdinal The ordinal position of the *Column* component to move.

newOrdinal The ordinal position where the *Column* should be moved to.

postAllDataSets()

```
public final void postAllDataSets()
```

Attempts to post any unposted rows in the *DataSet* and *DataSetView* components that share the same *StorageDataSet* property. On error, this method throws a *DataSetException*.

Overrides com.borland.dx.dataset.DataSet.postAllDataSets()

provideMoreData()

```
public boolean provideMoreData()
```

If a *Provider*, such as *QueryProvider* or *ProcedureProvider*, has a *load* property setting of *Load.AS_NEEDED*, calling this method retrieves the next batch of rows.

recalc()

```
public final synchronized void recalc()
```

Forces a recalculation of any calculated columns in the *StorageDataSet*.

refresh()

```
public void refresh()
```

Calls the *Provider* to refresh data from the data source of the *StorageDataSet*.

Overrides com.borland.dx.dataset.DataSet.refresh()

refreshSupported()

public boolean refreshSupported()

Returns **true** if the data source of the *StorageDataSet* supports refresh operations. Otherwise, this method returns **false**.

Overrides com.borland.dx.dataset.DataSet.refreshSupported()

removeLoadRowListener(listener)

public synchronized void removeLoadRowListener(LoadRowListener listener)

Removes the specified *LoadRowListener* object.

reset()

public final void reset()

Resets the *StorageDataSet* with the original values it was loaded with. All insert, update, and delete operations performed by the application are backed out.

resetPendingStatus(boolean)

public final void resetPendingStatus(boolean resolved)

Resets the status bits of the rows marked pending during resolution.

resolved

true if changes were resolved and the changed rows should now be treated as originals in a new resolution query.

false if changes were rolled back and the changed rows should still be treated as changed rows.

Overrides com.borland.dx.dataset.DataSet.resetPendingStatus(boolean)

resetPendingStatus(long, boolean)

public final void resetPendingStatus(long internalRow, boolean resolved)

Reset the status bits of a specific row.

internalRow

The row where the status bits should be reset.

resolved

true if changes were resolved and the changed rows should now be treated as originals in a new resolution query.

false if changes were rolled back and the changed rows should still be treated as changed rows.

Overrides com.borland.dx.dataset.DataSet.resetPendingStatus(long, boolean)

restructure()

```
public final void restructure()
```

Restructures the *StorageDataSet*. Currently meaningful only for *DataStore* after a move, delete, add, or change column method call.

The *needsRestructure* property returns whether a restructure operation is pending. A *StorageDataSet* with pending structural changes can still be used, and moved columns can be read and written to. Deleted columns are not visible, inserted columns can be read but not written and changed data type columns can be read but not written to.

The *restructure()* method can also be used to repair or compact a *StorageDataSet* and its associated indexes even when there are no pending structural changes.

saveChanges(com.borland.dx.dataset.DataSet)

```
public void saveChanges(DataSet dataSet)
```

Saves changes made to the data in the *StorageDataSet* back to its data source. If the *resolver* property is **null**, a *DataSetException* of *CANNOT_SAVE_CHANGES* is thrown..

dataSet The *DataSet* containing the updated data.

saveChangesSupported()

```
public boolean saveChangesSupported()
```

Returns **true** if the data source supports resolving changes made to the *StorageDataSet*, **false** if not. Typically, file-based data sources do not support data resolution.

Overrides com.borland.dx.dataset.DataSet.saveChangesSupported()

setRowId(java.lang.String, boolean)

```
public final synchronized void setRowId(String columnName, boolean setting)
```

Specifies that the named *Column* component either uniquely identifies a row in the server table where changes made to this *StorageDataSet* will be saved back to, or is one of a group of *Column* components that uniquely identifies a row. A **false** value indicates that the specified *Column* is not (part of) a unique row identifier.

columnName The String name of the *Column* component.

setting A boolean value indicating the participation of the *columnName* in the row identifier (**true**) or not (**false**).

startLoading(com.borland.dx.dataset.LoadCancel, int, boolean)

```
public final Variant[] startLoading(LoadCancel loader, int loadStatus, boolean loadAsync)
```

This method calls *startLoading(loader, loadStatus, loadAsync, false, false)*, where the last two parameters (*loadRowByRow* and *loadValidate*) are set to **false**. The following description is for the *startLoading(loader, loadStatus, loadAsync, loadRowbyRow, loadValidate)* method, which was added to late to make it into the documentation.

This method is used for high-speed loading of data into a *StorageDataSet*. It returns an array of *Variant* objects for all columns in a *DataSet*. You set the values in the array and call *loadRow()* or *loadRow(int)*. When the load operation is complete, you must call *endLoading*. Only one load operation may be active at one time.

This method may generate a *DataSetException* of *LOADING_NOT_STARTED*.

<i>loader</i>	A call back interface that requests the load operation to be canceled. Cannot be null .
<i>loadStatus</i>	Boolean that determines the status of the load process. Can be <i>RowStatus.INSERTED</i> or <i>RowStatus.LOADED</i> . Normally <i>RowStatus.LOADED</i> is used to indicate that the row was not inserted or updated.
<i>loadAsync</i>	Whether the loading should be done in a separate thread. Set this parameter to true if the intention is to load the <i>StorageDataSet</i> using a separate thread. This will cause periodic update notifications to be sent to any data aware controls and <i>StatusListeners</i> .
<i>loadRowByRow</i>	Boolean that indicates how to load rows. True means to load one row at a time. Use this with <i>QueryProviders</i> and <i>ProcedureProviders</i> that use the <i>Load.UNCACHED</i> option.
<i>loadValidate</i>	Boolean that indicates whether to have rows validated by calling <i>DataRow.validate()</i> for each row before it is loaded.

See also *loadRow(),endLoading()*

startLoading(com.borland.dx.dataset.LoadCancel, int, boolean, boolean)

```
public final synchronized Variant[] startLoading(LoadCancel loader, int loadStatus, boolean
    loadAsync, boolean loadUncached)
```

Calls *startLoading(loader, loadStatus, loadAsync, loadRowByRow, false)*. The last parameter, *loadValidate*, is set to **false**.

startLoading(com.borland.dx.dataset.LoadCancel, int, boolean, boolean, boolean)

```
public final synchronized Variant[] startLoading(LoadCancel loader, int loadStatus, boolean
    loadAsync, boolean loadUncached, boolean loadValidate)
```

Used for high speed loading of data into a *StorageDataSet*.

<i>loader</i>	A call back interface that requests the load operation to be canceled. Cannot be null .
<i>loadStatus</i>	Boolean that determines the status of the load process. Can be RowStatus.INSERTED or RowStatus.LOADED. Normally RowStatus.LOADED is used to indicate that the row was not inserted or loaded.
<i>loadAsync</i>	Whether the loading should be done in a separate thread. Set this parameter to true if the intention is to load the <i>StorageDataSet</i> using a separate thread. This will cause periodic update notifications to be sent to any data aware controls and <i>StatusListeners</i> .
<i>loadRowByRow</i>	Boolean that indicates how to load rows. True means to load one row at a time. Use this with <i>QueryProviders</i> and <i>ProcedureProviders</i> that use the Load.UNCACHED option.
<i>loadValidate</i>	Boolean that indicates whether to have rows validated by calling <i>DataRow.validate()</i> for each row before it is loaded.

StorageDataSet event listeners

This component is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

calcAggFields

```
public synchronized void addCalcAggFieldsListener(CalcAggFieldsListener listener)
public synchronized void removeCalcAggFieldsListener(CalcAggFieldsListener listener)
```

calcFields

```
public synchronized void addCalcFieldsListener(CalcFieldsListener listener)
public synchronized void removeCalcFieldsListener(CalcFieldsListener listener)
```

columnChange

```
public void addColumnChangeListener(ColumnChangeListener listener)
public synchronized void removeColumnChangeListener(ColumnChangeListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

edit

```
public void addEditListener(EditListener listener)
public synchronized void removeEditListener(EditListener listener)
```

load

```
public final synchronized void addLoadListener(LoadListener listener)
public final synchronized void removeLoadListener(LoadListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

StoreClassFactory interface

dx.dataset package

Classes that implement *StoreClassFactory* can be used to set the *StorageDataSet.storeClassFactory* property. Currently, this property is only meaningful when the *StorageDataSet.store* property is set to a *DataStoreConnection* or *DataStore* component.

The *StorageDataSet.storeClassFactory* property allows a *DataStore* table to be accessed only by a specific *StorageDataSet* class or *StorageDataSet* class extension. This allows an application to make sure that property and event settings are always activated when a *DataStore* table is accessed.

One useful example of how this mechanism can be used is the specification of Java triggers for *DataStore* tables. By setting the *StorageDataSet.storeClassFactory* property and the *EditListener* events of a *StorageDataSet*, Java trigger events can be enforced for both DataExpress and SQL access to a *DataStore* table.

When the *StorageDataSet.storeClassFactory* property is set, the name of the class that implements this interface is recorded in the metadata for the associated table in the *DataStore*. For DataExpress table opens, this property setting acts as a constraint. The *DataStore* will not allow write access to a table that has a *StoreClassFactory* metadata setting unless the *StorageDataSet* has its *storeClassFactory* property set to a class with the same name.

When a *DataStore* JDBC connection opens a table that has a *StoreClassFactory* recorded in its metadata, *DataStore* will do the following:

- 1 If the connection has never instantiated the *StoreClassFactory*, it will instantiate the *StoreClassFactory* and register this instance with the connection. (There will be one *StoreClassFactory* instantiation for each connection.)
- 2 *DataStore* will then call the *StoreClassFactory.getStorageDataSet()* method. The returned *StorageDataSet* will be used for the query operation. If the *StoreClassFactory* class cannot be instantiated, then write access will be denied to this table.

StoreClassFactory methods

Method	Implemented in
<code>getStorageDataSet(com.borland.dx.dataset.Store, java.lang.String)</code>	this class

getStorageDataSet(com.borland.dx.dataset.Store, java.lang.String)

StorageDataSet getStorageDataSet(Store store, String storeName)

Currently, this method is only called when a JDBC connection opens a table that has a *StoreClassFactory* recorded in its metadata.

This method returns a *StorageDataSet* for the given *storeName*.

store *Store* implementation that *storeName* is being opened for. Currently, this is always a *DataStoreConnection* used by a JDBC connection. This can be used to perform transactional operations with the same connection that is attempting to open a table.

storeName *storeName* of table that needs to be opened.

SumAggOperator class

dx.dataset package

Extends com.borland.dx.dataset.Aggregator

Implements java.io.Serializable, java.lang.Cloneable

The *SumAggOperator* class is an instantiable subclass of the *Aggregator* and defines a summary aggregation operation.

Set the *aggOperator* property of the *AggDescriptor* object to this class to perform a summary calculation using the property settings stored in the *AggDescriptor*. Attach the *AggDescriptor* object to a *Column* component's *agg* property to access the data that the aggregation uses.

SumAggOperator variables

Variable	Defined in
aggColumn	com.borland.dx.dataset.Aggregator
aggDataSet	com.borland.dx.dataset.Aggregator
aggValue	com.borland.dx.dataset.Aggregator
dataSet	com.borland.dx.dataset.Aggregator
resultColumn	com.borland.dx.dataset.Aggregator
resultValue	com.borland.dx.dataset.Aggregator

SumAggOperator properties

Property	Implemented in
class*	java.lang.Object

SumAggOperator methods

Method	Implemented in
add(com.borland.dx.dataset.ReadRow, long, boolean)	this class
clone()	com.borland.dx.dataset.AggOperator
delete(com.borland.dx.dataset.ReadRow, long)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
init(com.borland.dx.dataset.StorageDataSet, java.lang.String[], com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.Column, com.borland.dx.dataset.Column)	com.borland.dx.dataset.AggOperator
locate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.AggOperator
needsAggDataSet()	com.borland.dx.dataset.AggOperator
notify()	java.lang.Object
notifyAll()	java.lang.Object
open(com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.AggOperator
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add(com.borland.dx.dataset.ReadRow, long, boolean)

public void add(ReadRow row, long internalRow, boolean first)

A row has been added or updated.

- row*

The row containing the values.
- internalRow*

The unique identifier for the row.
- first*

Returns **true** if this is the first row in the group, **false** otherwise.

Overrides com.borland.dx.dataset.AggOperator.add(com.borland.dx.dataset.ReadRow, long, boolean)

TableDataSet component

Saving data in a *TextDataFile* to a JDBC data source

By default, data loaded into a *TableDataSet* using a *TextDataFile* is loaded with a *RowStatus.LOADED* status. Calling the *saveChanges(...)* method on a *QueryDataSet* or *ProcedureDataSet* has no effect because these rows are not considered as being inserted. Setting the *TextDataFile* property *setLoadAsInserted(true)* causes all rows loaded from the *TextDataFile* to be *RowStatus.INSERTED*. A subsequent call to *saveChanges(...)* with the *resolver* property set to a *QueryResolver* or *ProcedureResolver* will insert the rows into the JDBC data source for the *Resolver*.

Note This component will not delete existing rows if a *DataSet* already contains data.

TableDataSet constructors

TableDataSet()

public TableDataSet()

Instantiates a *TableDataSet* class object with default properties.

TableDataSet properties

Property	Implemented in
allRowIds**	com.borland.dx.dataset.StorageDataSet
assignedNull**	com.borland.dx.dataset.ReadWriteRow
calcAggFieldsListener*	com.borland.dx.dataset.StorageDataSet
calcFieldsListener*	com.borland.dx.dataset.StorageDataSet
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns**	com.borland.dx.dataset.StorageDataSet
dataFile	com.borland.dx.dataset.StorageDataSet
defaultValues**	com.borland.dx.dataset.DataSet
deletedRowCount*	com.borland.dx.dataset.StorageDataSet
detailDataSetWithFetchAsNeeded*	com.borland.dx.dataset.DataSet
details*	com.borland.dx.dataset.DataSet
displayErrors	com.borland.dx.dataset.DataSet
duplicates*	com.borland.dx.dataset.StorageDataSet
editable	com.borland.dx.dataset.DataSet
editing*	com.borland.dx.dataset.DataSet
editingNewRow*	com.borland.dx.dataset.DataSet
empty*	com.borland.dx.dataset.DataSet
enableDelete	com.borland.dx.dataset.DataSet

Property	Implemented in
enableInsert	com.borland.dx.dataset.DataSet
enableUpdate	com.borland.dx.dataset.DataSet
insertedRowCount*	com.borland.dx.dataset.StorageDataSet
internalRow*	com.borland.dx.dataset.DataSet
lastColumnVisited	com.borland.dx.dataset.DataSet
locale	com.borland.dx.dataset.StorageDataSet
masterLink	com.borland.dx.dataset.DataSet
maxDesignRows	com.borland.dx.dataset.StorageDataSet
maxResolveErrors	com.borland.dx.dataset.StorageDataSet
maxRows	com.borland.dx.dataset.StorageDataSet
metaDataUpdate	com.borland.dx.dataset.StorageDataSet
needsRestructure*	com.borland.dx.dataset.StorageDataSet
open*	com.borland.dx.dataset.DataSet
provider	com.borland.dx.dataset.StorageDataSet
readOnly	com.borland.dx.dataset.StorageDataSet
resolvable	com.borland.dx.dataset.StorageDataSet
resolveOrder	com.borland.dx.dataset.StorageDataSet
resolver	com.borland.dx.dataset.StorageDataSet
row*	com.borland.dx.dataset.DataSet
rowCount*	com.borland.dx.dataset.DataSet
rowFilterListener*	com.borland.dx.dataset.DataSet
schemaName	com.borland.dx.dataset.StorageDataSet
sort	com.borland.dx.dataset.DataSet
status*	com.borland.dx.dataset.DataSet
storageDataSet*	com.borland.dx.dataset.DataSet
store	com.borland.dx.dataset.StorageDataSet
storeClassFactory	com.borland.dx.dataset.StorageDataSet
storeName	com.borland.dx.dataset.StorageDataSet
tableName	com.borland.dx.dataset.StorageDataSet
unassignedNull**	com.borland.dx.dataset.ReadWriteRow
updatedRowCount*	com.borland.dx.dataset.StorageDataSet

TableDataSet methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	com.borland.dx.dataset.DataSet
addColumn(com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
addColumn(java.lang.String, int)	com.borland.dx.dataset.StorageDataSet

Method	Implemented in
addColumn(java.lang.String, java.lang.String, int)	com.borland.dx.dataset.StorageDataSet
addLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
addRow(com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
addUniqueColumn(com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
allocateValues()	com.borland.dx.dataset.DataSet
atFirst()	com.borland.dx.dataset.DataSet
atLast()	com.borland.dx.dataset.DataSet
cancel()	com.borland.dx.dataset.DataSet
cancelLoading()	com.borland.dx.dataset.StorageDataSet
cancelOperation()	com.borland.dx.dataset.StorageDataSet
canNavigate (com.borland.dx.dataset.Column, int)	com.borland.dx.dataset.DataSet
canSet(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
changeColumn(int, com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
changesPending()	com.borland.dx.dataset.StorageDataSet
clearStatus()	com.borland.dx.dataset.DataSet
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneColumns()	com.borland.dx.dataset.StorageDataSet
cloneDataSetStructure()	com.borland.dx.dataset.StorageDataSet
cloneDataSetView()	com.borland.dx.dataset.DataSet
close()	com.borland.dx.dataset.DataSet
closeProvider(boolean)	com.borland.dx.dataset.StorageDataSet
columnIsVisible(java.lang.String)	com.borland.dx.dataset.DataSet
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	com.borland.dx.dataset.DataSet
deleteDuplicates()	com.borland.dx.dataset.StorageDataSet
deleteRow()	com.borland.dx.dataset.DataSet
dittoRow(boolean, boolean)	com.borland.dx.dataset.DataSet
dittoRow(boolean)	com.borland.dx.dataset.DataSet
dropAllIndexes()	com.borland.dx.dataset.StorageDataSet
dropColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
dropColumn(java.lang.String)	com.borland.dx.dataset.StorageDataSet
dropIndex()	com.borland.dx.dataset.DataSet

Method	Implemented in
dropIndex (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
editRow()	com.borland.dx.dataset.DataSet
empty()	com.borland.dx.dataset.StorageDataSet
emptyAllRows()	com.borland.dx.dataset.DataSet
emptyRow()	com.borland.dx.dataset.DataSet
enableDataSetEvents(boolean)	com.borland.dx.dataset.DataSet
endLoading()	com.borland.dx.dataset.StorageDataSet
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	com.borland.dx.dataset.DataSet
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDataRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDataRow(int, com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDeletedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getDetail(java.lang.String)	com.borland.dx.dataset.DataSet
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet

Method	Implemented in
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInsertedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getOriginalRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.StorageDataSet
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getUpdatedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
goToClosestRow(int)	com.borland.dx.dataset.DataSet
goToInternalRow(long)	com.borland.dx.dataset.DataSet
goToRow(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
goToRow(int)	com.borland.dx.dataset.DataSet
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	com.borland.dx.dataset.DataSet
hashCode()	java.lang.Object
hasRowIds()	com.borland.dx.dataset.StorageDataSet

Method	Implemented in
hasValidations()	com.borland.dx.dataset.DataSet
inBounds()	com.borland.dx.dataset.DataSet
indexExists (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
insertRow(boolean)	com.borland.dx.dataset.DataSet
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	com.borland.dx.dataset.DataSet
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.DataSet
isModified(java.lang.String)	com.borland.dx.dataset.DataSet
isNew(int)	com.borland.dx.dataset.DataSet
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	com.borland.dx.dataset.DataSet
loadRow()	com.borland.dx.dataset.StorageDataSet
loadRow(int)	com.borland.dx.dataset.StorageDataSet
locate(com.borland.dx.dataset.ReadRow, int)	com.borland.dx.dataset.DataSet
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	com.borland.dx.dataset.DataSet
masterNavigated(com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
masterNavigating(com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
moveColumn(int, int)	com.borland.dx.dataset.StorageDataSet
moveRow(int)	com.borland.dx.dataset.DataSet
next()	com.borland.dx.dataset.DataSet
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.dx.dataset.DataSet
openDetails()	com.borland.dx.dataset.DataSet
post()	com.borland.dx.dataset.DataSet
postAllDataSets()	com.borland.dx.dataset.StorageDataSet
prior()	com.borland.dx.dataset.DataSet
provideMoreData()	com.borland.dx.dataset.StorageDataSet
recalc()	com.borland.dx.dataset.StorageDataSet
refetchRow (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.DataSet

Method	Implemented in
refilter()	com.borland.dx.dataset.DataSet
refresh()	com.borland.dx.dataset.StorageDataSet
refreshSupported()	com.borland.dx.dataset.StorageDataSet
removeLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
reset()	com.borland.dx.dataset.StorageDataSet
resetInBounds()	com.borland.dx.dataset.DataSet
resetPendingStatus(boolean)	com.borland.dx.dataset.StorageDataSet
resetPendingStatus(long, boolean)	com.borland.dx.dataset.StorageDataSet
restructure()	com.borland.dx.dataset.StorageDataSet
saveChanges()	com.borland.dx.dataset.DataSet
saveChanges (com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.StorageDataSet
saveChangesSupported()	com.borland.dx.dataset.StorageDataSet
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.DataSet
setDisplayVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setRowId(java.lang.String, boolean)	com.borland.dx.dataset.StorageDataSet
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startEditCheck (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean, boolean)	com.borland.dx.dataset.StorageDataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean)	com.borland.dx.dataset.StorageDataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean)	com.borland.dx.dataset.StorageDataSet
statusMessage (com.borland.dx.dataset.StatusEvent)	com.borland.dx.dataset.DataSet
statusMessage(int, java.lang.String)	com.borland.dx.dataset.DataSet
toggleViewOrder(java.lang.String)	com.borland.dx.dataset.DataSet
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
validate()	com.borland.dx.dataset.DataSet
validate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

TableDataSet event listeners

This component is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

calcAggFields

```
public synchronized void addCalcAggFieldsListener(CalcAggFieldsListener listener)
public synchronized void removeCalcAggFieldsListener(CalcAggFieldsListener listener)
```

calcFields

```
public synchronized void addCalcFieldsListener(CalcFieldsListener listener)
public synchronized void removeCalcFieldsListener(CalcFieldsListener listener)
```

columnChange

```
public void addColumnChangeListener(ColumnChangeListener listener)
public synchronized void removeColumnChangeListener(ColumnChangeListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

edit

```
public void addEditListener(EditListener listener)
public synchronized void removeEditListener(EditListener listener)
```

load

```
public final synchronized void addLoadListener(LoadListener listener)
public final synchronized void removeLoadListener(LoadListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

TextDataFile component

dx.dataset package

Extends com.borland.dx.dataset.DataFile**Implements** com.borland.dx.dataset.Designable, com.borland.dx.dataset.LoadCancel, java.io.Serializable

The *TextDataFile* component specifies the properties of a text file that affect its import and export, such as delimiters, field separators, and so on. This component is used when:

- importing data stored in a text format into a *TableDataSet* component
- exporting the data stored in any *StorageDataSet* to a text file

When importing data into a *TableDataSet*, this component specifies the properties of the text file that affect its import, such as delimiters, field separators, and so on. To further specify the formatting of the data within each field, set the *exportDisplayMask* property of the *Column* component. The *exportDisplayMask* is used both when importing as well as exporting.

This component is the default for the *dataFile* property of the *StorageDataSet* component. By default, exported data is in a text format as specified by this component's properties. To write the data stored in any *StorageDataSet* to a text file, instantiate a *TextDataFile* component and call one of the *TextDataFile.save(...)* methods.

All properties of this component have default values. To change these values, call the corresponding accessor methods. In addition, localized properties are stored in its associated *DataFileFormat* object, which is set by this component's *fileFormat* property.

For information on saving data stored in a *TableDataSet* using this component, see "Saving data in a *TextDataFile* to a JDBC data source" in the About section of the *TableDataSet* component.

TextDataFile constructors

TextDataFile()

public TextDataFile()

Constructs a *TextDataFile* component.

TextDataFile properties

Property	Implemented in
class*	java.lang.Object
delimiter	this class
encoding	this class
fileFormat	this class
fileName	this class
loadAsInserted	this class
loadOnOpen	this class
locale	this class
separator	this class

delimiter

public final String getDelimiter()
public final void setDelimiter(String delimiter)

The *String* character that appears before and after character data elements in the data file. The default value is double-quotes (“”).

encoding

public final String getEncoding()
public final void setEncoding(String encoding)

The encoding of the file. Based on the encoding returned by *System.getProperty(“file.encoding”)*, this property sets the *fileFormat* property to *ASCII* or *ENCODED*.

fileFormat

```
public final int getFileFormat()
public final void setFileFormat(int fileFormat)
```

The format of the file-based data source. This property defaults to *ASCII*. Valid values for *encoding* are defined in *DataFileFormat*.

fileName

```
public String getFileName()
public void setFileName(String fileName)
```

Stores the name of the file that contains data to read in or to write out to. This property defaults to *TextDataFile.txt*.

loadAsInserted

```
public final boolean isLoadAsInserted()
public final void setLoadAsInserted(boolean loadAsInserted)
```

Determines whether to load rows with a *RowStatus* of *RowStatus.INSERTED* (**true**) or, to load rows with a *RowStatus* of *RowStatus.LOADED* (**false**). If set to *true* these rows are treated as inserted when methods such as *Database.saveChanges()* are called.

loadOnOpen

```
public boolean isLoadOnOpen()
public void setLoadOnOpen(boolean loadOnOpen)
```

If **true**, then the *DataSet* is loaded with the contents of the file when the *DataSet* is opened.

locale

```
public final Locale getLocale()
public final void setLocale(Locale locale)
```

Specifies the *Locale* of the *TextDataFile* component. The *locale* property allows the user to identify which locale to use when formatting the data in a *Column*. This property supports locales that are supported by the JavaSoft™ JDK.

separator

```
public final String getSeparator()
public final void setSeparator(String separator)
```

Specifies the value used to separated individual data items. The default value for this property is the tab character (`\t`).

TextDataFile methods

Method	Implemented in
cancelLoad()	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
load(com.borland.dx.dataset.DataSet, java.io.InputStream, java.io.InputStream)	this class
load(com.borland.dx.dataset.DataSet)	this class
loadMetaData(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
save(com.borland.dx.dataset.DataSet, java.io.OutputStream, java.io.OutputStream)	this class
save(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

load(com.borland.dx.dataset.DataSet)

public final void load(DataSet dataSet)

Loads data from a stream into the *DataSet*.

Overrides com.borland.dx.dataset.DataFile.load(com.borland.dx.dataset.DataSet)

load(com.borland.dx.dataset.DataSet, java.io.InputStream, java.io.InputStream)

public final void load(DataSet dataSet, InputStream stream, InputStream schemaStream)

Loads data from the input stream into the specified *DataSet*. The *schemaStream* parameter can be set to null if columns have already been added to the *DataSet*.

loadMetaData(com.borland.dx.dataset.DataSet)

public final void loadMetaData(DataSet dataSet)

Loads metadata into the *DataSet* using the schema file specified by the *fileName* property.

Overrides com.borland.dx.dataset.DataFile.loadMetaData(com.borland.dx.dataset.DataSet)

save(com.borland.dx.dataset.DataSet)

```
public void save(DataSet saveDataSet)
```

Saves *DataSet* data to the file specified by the *fileName* property.

Overrides `com.borland.dx.dataset.DataFile.save(com.borland.dx.dataset.DataSet)`

save(com.borland.dx.dataset.DataSet, java.io.OutputStream, java.io.OutputStream)

```
public void save(DataSet saveDataSet, OutputStream stream, OutputStream schemaStream)
```

Saves the *DataSet* data and metadata as specified.

saveDataSet

The *DataSet* containing the data to save.

stream

The stream that will contain the actual data from the *DataSet*.

schemaStream

The stream that will contain the *DataSet's* metadata. If this parameter is null, the *fileName* property is used to save the metadata.

UpdateMode interface

dx.dataset package

The *UpdateMode* component collects constants used when resolving changes for the *QueryDataSet* and *ProcedureDataSet* components.

UpdateMode variables

Variable	Defined in
ALL_COLUMNS	this class
CHANGED_COLUMNS	this class
KEY_COLUMNS	this class
UNASSIGNED	this class

ALL_COLUMNS

```
public static final int ALL_COLUMNS = 1
```

Every column is used to find the row being updated (the default). This is the most restrictive mode.

CHANGED_COLUMNS

public static final int CHANGED_COLUMNS = 3

The key columns and columns that have changed are used to find the record being updated.

KEY_COLUMNS

public static final int KEY_COLUMNS = 2

Only the key columns are used to find the record being updated. This is the least restrictive mode and should be used only if other users will not be changing the records being updated.

UNASSIGNED

public static final int UNASSIGNED = 0

The update mode has not been assigned.

ValidationException class

dx.dataset package

Extends com.borland.dx.dataset.DataSetException

Implements com.borland.jb.util.ChainedException, java.io.Serializable

The *ValidationException* class is a subclass of *DataSetException* and is used heavily by the *dataset* package for *Column* and row-level validation errors that occur when posting changed or new row(s) of data.

ValidationException variables

Variable	Defined in
ALREADY_LOADING	com.borland.dx.dataset.DataSetException
APPLICATION_ERROR	this class
BAD_PROCEDURE_PROPERTIES	com.borland.dx.dataset.DataSetException
BAD_QUERY_PROPERTIES	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN_DATA_TYPE	com.borland.dx.dataset.DataSetException
CANNOT_DITTO_EXISTING	this class
CANNOT_FIND_TABLE_NAME	com.borland.dx.dataset.DataSetException
CANNOT_IMPORT_NULL_DATASET	com.borland.dx.dataset.DataSetException
CANNOT_ORPHAN_DETAILS	this class

Variable	Defined in
CANNOT_REFRESH	com.borland.dx.dataset.DataSetException
CANNOT_SAVE_CHANGES	com.borland.dx.dataset.DataSetException
CANNOT_UPDATE_SCOPED_DATA_ROW	com.borland.dx.dataset.DataSetException
CLASS_NOT_FOUND_ERROR	com.borland.dx.dataset.DataSetException
COLUMN_ALREADY_BOUND	com.borland.dx.dataset.DataSetException
COLUMN_NOT_IN_ROW	com.borland.dx.dataset.DataSetException
COLUMN_TYPE_CONFLICT	com.borland.dx.dataset.DataSetException
CONNECTION_DESCRIPTOR_NOT_SET	com.borland.dx.dataset.DataSetException
CONNECTION_NOT_CLOSED	com.borland.dx.dataset.DataSetException
DATA_FILE_LOAD_FAILED	com.borland.dx.dataset.DataSetException
DATASET_CORRUPT	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_ROWS	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_TABLES	com.borland.dx.dataset.DataSetException
DATASET_NOT_OPEN	com.borland.dx.dataset.DataSetException
DATASET_OPEN	com.borland.dx.dataset.DataSetException
DELETE_DUPLICATES	com.borland.dx.dataset.DataSetException
DELETE_NOT_ALLOWED	this class
DRIVER_NOT_LOADED_AT_RUNTIME	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_IN_DESIGN	com.borland.dx.dataset.DataSetException
DUPLICATE_COLUMN_NAME	com.borland.dx.dataset.DataSetException
DUPLICATE_KEY	this class
DUPLICATE_PRIMARY	com.borland.dx.dataset.DataSetException
EMPTY_COLUMN_NAMES	com.borland.dx.dataset.DataSetException
errorCode	com.borland.dx.dataset.DataSetException
EXCEPTION_CHAIN	com.borland.dx.dataset.DataSetException
exceptionChain	com.borland.dx.dataset.DataSetException
FIELD_POST_ERROR	com.borland.dx.dataset.DataSetException
GENERIC_ERROR	com.borland.dx.dataset.DataSetException
GREATER_THAN_MAX	this class
INCOMPATIBLE_DATA_ROW	com.borland.dx.dataset.DataSetException
INSERT_NOT_ALLOWED	this class
INSUFFICIENT_ROWID	com.borland.dx.dataset.DataSetException
INVALID_AGG_DESCRIPTOR	com.borland.dx.dataset.DataSetException
INVALID_CLASS	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_POSITION	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_TYPE	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_VALUE	this class
INVALID_DATA_FILE_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_FORMAT	this class
INVALID_ITERATOR_USE	com.borland.dx.dataset.DataSetException

Variable	Defined in
INVALID_PRECISION	this class
INVALID_ROW_VALUES	this class
INVALID_SCHEMA_FILE	com.borland.dx.dataset.DataSetException
INVALID_SORT_COLUMN	com.borland.dx.dataset.DataSetException
INVALID_STORE_CLASS	com.borland.dx.dataset.DataSetException
INVALID_STORE_NAME	com.borland.dx.dataset.DataSetException
IO_ERROR	com.borland.dx.dataset.DataSetException
LESS_THAN_MIN	this class
LINK_COLUMNS_ERROR	com.borland.dx.dataset.DataSetException
LINKFIELD_IN_USERPARAMETERS	com.borland.dx.dataset.DataSetException
LOADING_NOT_STARTED	com.borland.dx.dataset.DataSetException
MASTER_DETAIL_VIEW_ERROR	com.borland.dx.dataset.DataSetException
MASTER_NAVIGATION_ERROR	com.borland.dx.dataset.DataSetException
MISMATCH_PARAM_RESULT	com.borland.dx.dataset.DataSetException
MISMATCHED_PARAMETER_FORMAT	com.borland.dx.dataset.DataSetException
MISSING_MASTER_DATASET	com.borland.dx.dataset.DataSetException
MISSING_REPLACESTOREROW	com.borland.dx.dataset.DataSetException
MISSING_RESOLVER	com.borland.dx.dataset.DataSetException
MULTIPLE_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NEED_LOCATE_START_OPTION	com.borland.dx.dataset.DataSetException
NEED_PROCEDUREPROVIDER	com.borland.dx.dataset.DataSetException
NEED_QUERYPROVIDER	com.borland.dx.dataset.DataSetException
NEED_STORAGEDATASET	com.borland.dx.dataset.DataSetException
NEEDS_RECALC	com.borland.dx.dataset.DataSetException
NO_CALC_AGG_FIELDS	com.borland.dx.dataset.DataSetException
NO_CALC_FIELDS	com.borland.dx.dataset.DataSetException
NO_DATABASE_TO_RESOLVE	com.borland.dx.dataset.DataSetException
NO_NON_BLOB_COLUMNS	com.borland.dx.dataset.DataSetException
NO_PRIMARY_KEY	com.borland.dx.dataset.DataSetException
NO_PRIOR_ORIGINAL_ROW	com.borland.dx.dataset.DataSetException
NO_RESULT_SET	com.borland.dx.dataset.DataSetException
NO_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NO_ROWS_TO_DELETE	this class
NO_UPDATABLE_COLUMNS	com.borland.dx.dataset.DataSetException
NO_WHERE_CLAUSE	com.borland.dx.dataset.DataSetException
NON_EXISTENT_ROWID	com.borland.dx.dataset.DataSetException
NOT_DATABASE_RESOLVER	com.borland.dx.dataset.DataSetException
NOT_SELECT_QUERY	com.borland.dx.dataset.DataSetException
NOT_UPDATEABLE	com.borland.dx.dataset.DataSetException
NULL_COLUMN_NAME	com.borland.dx.dataset.DataSetException
ONEPASS_INPUT_STREAM	com.borland.dx.dataset.DataSetException

Variable	Defined in
PARAMETER_COUNT_MISMATCH	com.borland.dx.dataset.DataSetException
PARTIAL_SEARCH_FOR_STRING	com.borland.dx.dataset.DataSetException
PROCEDURE_FAILED	com.borland.dx.dataset.DataSetException
PROCEDURE_IN_PROCESS	com.borland.dx.dataset.DataSetException
PROVIDER_FAILED	com.borland.dx.dataset.DataSetException
PROVIDER_OWNED	com.borland.dx.dataset.DataSetException
QUERY_FAILED	com.borland.dx.dataset.DataSetException
QUERY_IN_PROCESS	com.borland.dx.dataset.DataSetException
READ_ONLY_COLUMN	this class
READ_ONLY_DATASET	this class
READ_ONLY_STORE	com.borland.dx.dataset.DataSetException
REFRESHROW_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
REOPEN_FAILURE	com.borland.dx.dataset.DataSetException
RESOLVE_FAILED	com.borland.dx.dataset.DataSetException
RESOLVE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
RESTRUCTURE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
SET_CALCULATED_FAILURE	com.borland.dx.dataset.DataSetException
SQL_ERROR	com.borland.dx.dataset.DataSetException
TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
UNEXPECTED_END_OF_QUERY	com.borland.dx.dataset.DataSetException
UNKNOWN_COLUMN_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_DETAIL_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_PARAM_NAME	com.borland.dx.dataset.DataSetException
UNRECOGNIZED_DATA_TYPE	com.borland.dx.dataset.DataSetException
UPDATE_NOT_ALLOWED	this class
URL_NOT_FOUND	com.borland.dx.dataset.DataSetException
URL_NOT_FOUND_IN_DESIGN	com.borland.dx.dataset.DataSetException
WRONG_DATABASE	com.borland.dx.dataset.DataSetException

APPLICATION_ERROR

```
public static final int APPLICATION_ERROR = BASE+12
```

Application error caused by an *Exception* in a application event handler.

CANNOT_DITTO_EXISTING

```
public static final int CANNOT_DITTO_EXISTING = BASE+10
```

Cannot ditto into an existing row. By default, you cannot ditto over an existing row; you must insert an empty row and ditto into it.

CANNOT_ORPHAN_DETAILS

```
public static final int CANNOT_ORPHAN_DETAILS = BASE+3
```

Master rows that have detail rows linked to them cannot be deleted or have their linking columns modified.

DELETE_NOT_ALLOWED

```
public static final int DELETE_NOT_ALLOWED = BASE+16
```

Row deleting not allowed.

DUPLICATE_KEY

```
public static final int DUPLICATE_KEY = BASE+17
```

The key value is a duplicate.

GREATER_THAN_MAX

```
public static final int GREATER_THAN_MAX = BASE+8
```

The value entered for a *Column* is greater than the *Column* component's *max* property.

INSERT_NOT_ALLOWED

```
public static final int INSERT_NOT_ALLOWED = BASE+14
```

Row insertion not allowed.

INVALID_COLUMN_VALUE

```
public static final int INVALID_COLUMN_VALUE = BASE+4
```

Application-defined validation failed in a *ColumnChangeListener.validate()* event handler.

INVALID_FORMAT

```
public static final int INVALID_FORMAT = BASE+11
```

Cannot parse value or generic format error.

Overrides `com.borland.dx.dataset.DataSetException.INVALID_FORMAT`

INVALID_PRECISION

```
public static final int INVALID_PRECISION = BASE+9
```

String values for this column can not exceed the precision length set in the *Column.precision* property. This property can be implicitly set when data is retrieved from a provider. If data is being provided from a JDBC driver, the precision specified by the JDBC result set will be propagated to the *Column.precision* property. This propagation can be overridden by explicitly setting the *Column.precision* property.

INVALID_ROW_VALUES

```
public static final int INVALID_ROW_VALUES = BASE+5
```

The row cannot be posted due to missing or invalid field values.

LESS_THAN_MIN

```
public static final int LESS_THAN_MIN = BASE+7
```

The value entered for the *Column* is less than the *Column* component's *min* property.

NO_ROWS_TO_DELETE

```
public static final int NO_ROWS_TO_DELETE = BASE+13
```

No rows to delete.

READ_ONLY_COLUMN

```
public static final int READ_ONLY_COLUMN = BASE+1
```

Attempting to assign a value to a *readonly* column.

READ_ONLY_DATASET

```
public static final int READ_ONLY_DATASET = BASE+2
```

Attempting to assign a value to a *readonly DataSet*.

UPDATE_NOT_ALLOWED

```
public static final int UPDATE_NOT_ALLOWED = BASE+15
```

Row editing not allowed.

ValidationException constructors

ValidationException(int, java.lang.String, com.borland.dx.dataset.Column)

public ValidationException(int errorCode, String error, Column column)

Constructs a *ValidationException* object that contains the error code, the error that occurred, and the *Column* in which it occurred.

<i>errorCode</i>	One of the <i>ValidationException</i> variables.
<i>error</i>	The message describing the error that occurred.
<i>column</i>	The column containing the error that occurred.

ValidationException(int, java.lang.String, com.borland.dx.dataset.Column, java.lang.Exception)

public ValidationException(int errorCode, String error, Column column, Exception ex)

Constructs a *ValidationException* object that contains the error code, the error that occurred, the *Column* in which it occurred, and the exception.

<i>errorCode</i>	One of the <i>ValidationException</i> variables.
<i>error</i>	The message describing the error that occurred.
<i>column</i>	The column containing the error that occurred.
<i>ex</i>	The exception that occurred.

ValidationException properties

Property	Implemented in
class*	java.lang.Object
errorCode*	com.borland.dx.dataset.DataSetException
errorColumn*	this class
exceptionChain*	com.borland.dx.dataset.DataSetException
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

errorColumn

public final Column getErrorColumn()

Returns the offending *Column* for this error. Returns **null** if it is a row level error.

ValidationException methods

Method	Implemented in
<code>addExceptionListener(com.borland.dx.dataset.ExceptionListener)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>badProcedureProperties()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>badQueryProperties()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>classNotFoundException</code> (<code>java.lang.ClassNotFoundException</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>clone()</code>	<code>java.lang.Object</code>
<code>connectionDescriptorNotSet()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>connectionNotClosed(java.lang.Exception)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>dataSetHasNoTable()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>dataSetNotOpen()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>deleteDuplicates()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>driverNotLoadedAtRuntime</code> (<code>java.lang.String</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>driverNotLoadedInDesign</code> (<code>java.lang.String</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>duplicateKey</code> (<code>com.borland.dx.dataset.StorageDataSet</code> , <code>com.borland.dx.dataset.SortDescriptor</code>)	this class
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>fillInStackTrace()</code>	<code>java.lang.Throwable</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>getExceptionListeners()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>insufficientRowId()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidClass(java.lang.Class)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidClass(java.lang.String, java.lang.String)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidColumnType</code> (<code>com.borland.dx.dataset.Column</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidFormat(java.lang.Exception, java.lang.String, java.lang.String)</code>	this class
<code>invalidSQLType(int)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidStoreName(java.lang.String)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>IOException(java.io.IOException)</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>mismatchedParameterFormat()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>mismatchParamResult()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>missingMasterDataSet()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>mkUrlNotFound(java.lang.String, java.lang.Exception)</code>	<code>com.borland.dx.dataset.DataSetException</code>

Method	Implemented in
mkUrlNotFoundInDesign (java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
multipleRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
needProcedureProvider()	com.borland.dx.dataset.DataSetException
needQueryProvider()	com.borland.dx.dataset.DataSetException
needsRecalc(java.lang.String)	com.borland.dx.dataset.DataSetException
noDatabaseOnResolver()	com.borland.dx.dataset.DataSetException
nonExistentRowId()	com.borland.dx.dataset.DataSetException
noResultSet()	com.borland.dx.dataset.DataSetException
noRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
notDatabaseResolver()	com.borland.dx.dataset.DataSetException
notify()	java.lang.Object
notifyAll()	java.lang.Object
notSelectQuery()	com.borland.dx.dataset.DataSetException
notSortable()	com.borland.dx.dataset.DataSetException
noUpdatableColumns()	com.borland.dx.dataset.DataSetException
noWhereClause (com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.DataSetException
onePassInputStream (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
parameterCountMismatch(int, int, int)	com.borland.dx.dataset.DataSetException
printStackTrace()	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintStream)	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
procedureFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
procedureInProcess()	com.borland.dx.dataset.DataSetException
providerFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
providerOwned()	com.borland.dx.dataset.DataSetException
queryFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
queryInProcess()	com.borland.dx.dataset.DataSetException
readOnlyStore(java.lang.String)	com.borland.dx.dataset.DataSetException
removeExceptionListener(com.borland. dx.dataset.ExceptionListener)	com.borland.dx.dataset.DataSetException
resolveFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
SQLException(java.sql.SQLException)	com.borland.dx.dataset.DataSetException
throwException(int, java.lang.Exception)	com.borland.dx.dataset.DataSetException
throwExceptionChain (java.lang.Throwable)	com.borland.dx.dataset.DataSetException
toString()	java.lang.Throwable
transactionIsolationLevelNotSupported()	com.borland.dx.dataset.DataSetException
unexpectedEndOfQuery()	com.borland.dx.dataset.DataSetException
unknownColumnName(java.lang.String)	com.borland.dx.dataset.DataSetException

Method	Implemented in
unknownDetailName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownParamName(java.lang.String)	com.borland.dx.dataset.DataSetException
unrecognizedDataType()	com.borland.dx.dataset.DataSetException
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
wrongDatabase()	com.borland.dx.dataset.DataSetException

duplicateKey(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.SortDescriptor)

public static final void duplicateKey(StorageDataSet dataSet, SortDescriptor descriptor)

Cannot parse value or generic format error. Throws a *ValidationException* of DUPLICATE_KEY.

invalidFormat(java.lang.Exception, java.lang.String, java.lang.String)

public static final void invalidFormat(Exception ex, String columnName, String message)

Cannot parse value or generic format error. If *message* is **null**, throws a *ValidationException* of INVALID_FORMAT. If *message* is not **null**, returns the offending column for this error, or returns **null** if it is a row level error.

Variant component

dx.dataset package

Extends	java.lang.Object	
Extended by	com.borland.dx.dataset.ColumnVariant, com.borland.dx.dataset.RowVariant	
Implements	java.io.Serializable, java.lang.Cloneable	
	The <i>Variant</i> class is a type of storage class whose value can be one of many data types. It can hold data of these types:	
	BigDecimal	boolean
	byte array	Date
	float	input stream
	long	Object
	String	Time
		byte
		double
		int
		short
		Timestamp

Variant contains constants used to identify all of these data types. It also contains the methods to get and set data values and to perform operations on *Variant* data, such as addition, subtraction, and comparing one value to another. The *dataset* package uses the *Variant* data type frequently because it can handle all types of data.

Variant variables

Variable	Defined in
ASSIGNED_NULL	this class
AssignedNull_S	this class
BIGDECIMAL	this class
BigDecimalType_S	this class
BinaryStreamType_S	this class
BOOLEAN	this class
BooleanType_S	this class
BYTE	this class
BYTE_ARRAY	this class
ByteArrayType_S	this class
ByteType_S	this class
DATE	this class
DateType_S	this class
DOUBLE	this class
DoubleType_S	this class
FLOAT	this class
FloatType_S	this class
INPUTSTREAM	this class
InputStreamType_S	this class
INT	this class
IntType_S	this class
LONG	this class
LongType_S	this class
MaxTypes	this class
NULL_TYPES	this class
nullVariant	this class
OBJECT	this class
ObjectType_S	this class
SHORT	this class
ShortType_S	this class
STRING	this class
StringType_S	this class
TIME	this class

Variable	Defined in
TIMESTAMP	this class
TimestampType_S	this class
TimeType_S	this class
UNASSIGNED_NULL	this class
UnassignedNull_S	this class
UnknownType_S	this class

ASSIGNED_NULL

```
public static final int ASSIGNED_NULL = 1
```

Constant that identifies a data type for values that are explicitly set to **null**. This is in contrast to data that is never assigned.

See also `UNASSIGNED_NULL`

AssignedNull_S

```
public static final String AssignedNull_S = "ASSIGNED_NULL"
```

A constant that displays an assigned null value as the string "ASSIGNED_NULL". An assigned null is a value explicitly set to **null** in contrast to one that is simply not assigned.

BIGDECIMAL

```
public static final int BIGDECIMAL = 10
```

An integer constant used to identify the *BigDecimal* data type. *BigDecimal* values have an unlimited precision integer value and an integer scale factor.

BigDecimalType_S

```
public static final String BigDecimalType_S = "BIGDECIMAL"
```

A constant that represents the *BigDecimal* data type as the string "BIGDECIMAL".

BinaryStreamType_S

```
public static final String BinaryStreamType_S = "BINARY_STREAM"
```

This constant was deprecated. Use `InputStreamType_S` instead.

BOOLEAN

```
public static final int BOOLEAN = 11
```

An integer constant used to identify data of type **boolean**.

BooleanType_S

```
public static final String BooleanType_S = "BOOLEAN"
```

A constant that represents the **boolean** data type as the string “BOOLEAN”.

BYTE

```
public static final int BYTE = 2
```

An integer constant used to identify data of type **byte**.

BYTE_ARRAY

```
public static final int BYTE_ARRAY = 18
```

An integer constant used to identify data in a **byte** array.

ByteArrayType_S

```
public static final String ByteArrayType_S = "BYTE_ARRAY"
```

A constant that displays a **byte** array as the string “BYTE_ARRAY”.

ByteType_S

```
public static final String ByteType_S = "BYTE"
```

A constant that represents the **byte** data type as the string “BYTE”.

DATE

```
public static final int DATE = 13
```

An integer constant used to identify the *Date* data type.

DateType_S

```
public static final String DateType_S = "DATE"
```

A constant that represents the *Date* data type as the string “DATE”.

DOUBLE

```
public static final int DOUBLE = 7
```

An integer constant used to identify the **double** data type.

DoubleType_S

```
public static final String DoubleType_S = "DOUBLE"
```

A constant that represents the **double** date type as the string “DOUBLE”.

FLOAT

```
public static final int FLOAT = 6
```

An integer constant used to identify the **float** data type.

FloatType_S

```
public static final String FloatType_S = "FLOAT"
```

A constant that represents the **float** data type as the string "FLOAT".

INPUTSTREAM

```
public static final int INPUTSTREAM = 12
```

An integer constant used to identify data of a input stream.

InputStreamType_S

```
public static final String InputStreamType_S = "INPUTSTREAM"
```

A constant that represents the INPUTSTREAM data type as the string "INPUTSTREAM".

INT

```
public static final int INT = 4
```

An integer constant used to identify the **int** data type.

IntType_S

```
public static final String IntType_S = "INT"
```

A constant that represents the **int** data type as the string "INT".

LONG

```
public static final int LONG = 5
```

An integer constant used to identify the **long** data type.

LongType_S

```
public static final String LongType_S = "LONG"
```

A constant that represents the **long** data type as the string "LONG".

MaxTypes

```
public static final int MaxTypes = 18
```

The maximum number of data types *Variant* can handle.

NULL_TYPES

```
public static final int NULL_TYPES = 1
```

An integer constant used to identify **null** data. Null data can be either assigned or unassigned. See ASSIGNED_NULL and UNASSIGNED_NULL.

nullVariant

```
public static final Variant nullVariant = new Variant(UNASSIGNED_NULL)
```

An integer constant used to identify a *Variant* data type with an unassigned null data value.

OBJECT

```
public static final int OBJECT = 17
```

An integer constant used to identify the *Object* data type.

ObjectType_S

```
public static final String ObjectType_S = "OBJECT"
```

A constant that represents the *Object* data type as the string "OBJECT".

SHORT

```
public static final int SHORT = 3
```

An integer constant used to identify the **short** data type.

ShortType_S

```
public static final String ShortType_S = "SHORT"
```

A constant that represents the **short** data type as the string "SHORT".

STRING

```
public static final int STRING = 16
```

An integer constant used to identify the *String* data type.

StringType_S

```
public static final String StringType_S = "STRING"
```

A constant that represents the *String* data type as the string “STRING”.

TIME

```
public static final int TIME = 14
```

An integer constant used to identify the *Time* data type.

TIMESTAMP

```
public static final int TIMESTAMP = 15
```

An integer constant used to identify the *TimeStamp* data type.

TimestampType_S

```
public static final String TimestampType_S = "TIMESTAMP"
```

A constant that represents the *TimeStamp* data type as the string “TIMESTAMP”.

TimeType_S

```
public static final String TimeType_S = "TIME"
```

A constant that represents the *Time* data type as the string “TIME”.

UNASSIGNED_NULL

```
public static final int UNASSIGNED_NULL = 0
```

An integer constant used to identify an unassigned null value. An unassigned null value is a data value that was never assigned. This is in contrast to an assigned null value that is explicitly assigned. See `ASSIGNED_NULL`.

UnassignedNull_S

```
public static final String UnassignedNull_S = "UNASSIGNED_NULL"
```

A constant that represents an unassigned null as the string “UNASSIGNED_NULL”.

UnknownType_S

```
public static final String UnknownType_S = "UNKNOWN"
```

A constant that represents an unknown data type as the string “UNKNOWN”.

Variant constructors

Variant()

public Variant()

Constructs a *Variant* object without specifying the explicit data type.

Variant(int)

public Variant(int dataType)

Constructs a *Variant* object that can contain data of the type specified with the *dataType* parameter. Variants instantiated with this constructor must have the results of all get or set operations be of the type specified.

dataType

If this constructor is used, all set operations must be of the data type from which the Variant was constructed.

Variant properties

Property	Implemented in
arrayLength	this class
asBigDecimal*	this class
asBoolean*	this class
asDate**	this class
asDouble*	this class
asFloat*	this class
asInt*	this class
asLong*	this class
asObject*	this class
asShort*	this class
assignedNull*	this class
asTime**	this class
asTimestamp**	this class
asVariant**	this class
bigDecimal	this class
binaryStream	this class
boolean	this class
byte	this class
byteArray*	this class
class*	java.lang.Object
date	this class

Property	Implemented in
displayValue*	this class
double	this class
float	this class
inputStream	this class
int	this class
long	this class
null	this class
object	this class
setType*	this class
short	this class
string	this class
time	this class
timestamp	this class
type*	this class
unassignedNull*	this class
variant**	this class

arrayLength

```
public final int getArrayLength()
public final void setArrayLength(int length)
```

Retrieves and sets the length of an array.

length The length of the array.

asBigDecimal

```
public final BigDecimal getAsBigDecimal()
```

Obtains a data value as a *BigDecimal* data type. It can handle data identified as type BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIME, DATE, TIMESTAMP, UNASSIGNED_NULL, and ASSIGNED_NULL.

asBoolean

```
public final boolean getAsBoolean()
```

Read-only property that obtains a data value as a **boolean** data type. It can handle data identified as type BOOLEAN, STRING, BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, TIME, DATE, and TIMESTAMP.

asDate

```
public final void setAsDate(Variant value)
```

Write-only property that sets the value of this *Variant* as a date or date and time value. It can set data identified as type DATE, BOOLEAN, BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIME, and TIMESTAMP.

asDouble

```
public final double getAsDouble()
```

Read-only property that retrieves the value of this *Variant* as a **double** data type. It can obtain data identified as type BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIME, DATE, TIMESTAMP, UNASSIGNED_NULL, and ASSIGNED_NULL.

asFloat

```
public final float getAsFloat()
```

Read-only property that retrieves the value of this *Variant* as a **float** data type. It can obtain data identified as type BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIME, DATE, TIMESTAMP, UNASSIGNED_NULL, and ASSIGNED_NULL.

asInt

```
public final int getAsInt()
```

Read-only property that retrieves the value of this *Variant* as a **int** data type. It can obtain data identified as type BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIME, DATE, TIMESTAMP, UNASSIGNED_NULL, and ASSIGNED_NULL.

asLong

```
public final long getAsLong()
```

Read-only property that retrieves the value of this *Variant* as a **long** data type. It can obtain data identified as type BOOLEAN, BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIMESTAMP, TIME, DATE, UNASSIGNED_NULL, and ASSIGNED_NULL.

asObject

```
public Object getAsObject()
```

Read-only property that retrieves the value of this *Variant* as an **object** data type.

asShort

```
public final short getAsShort()
```

Read-only property that retrieves the value of this *Variant* as a **short** data type. It can obtain data identified as type BYTE, SHORT, INT, BOOLEAN, LONG, FLOAT, DOUBLE, BIGDECIMAL, TIMESTAMP, TIME, DATE, UNASSIGNED_NULL, and ASSIGNED_NULL.

assignedNull

```
public final boolean isAssignedNull()
```

Read-only property that determines whether the data value is an assigned **null** value. If it returns true, the value is an assigned **null** value; otherwise, it is not.

asTime

```
public final void setAsTime(Variant value)
```

Sets Time to value.

value If *value* is of type TIME, the value is copied directly. If *value* is of type BOOLEAN, BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, DATE, TIMESTAMP, *setAsTime* is called with the return value from *value.getAsLong()*. If *value* is of type ASSIGNED_NULL or UNASSIGNED_NULL, Time is set to the same *_NULL value.

asTimestamp

```
public final void setAsTimestamp(Variant value)
```

Sets Timestamp to value.

value If *value* is of type TIME, the value is copied directly. If *value* is of type BOOLEAN, BYTE, SHORT, INT, LONG, FLOAT, DOUBLE, BIGDECIMAL, DATE, TIMESTAMP, *setTimeStamp()* is called with the return value from *value.getAsLong()*. If *value* is of type ASSIGNED_NULL or UNASSIGNED_NULL, Time is set to the same *_NULL value.

asVariant

```
public final void setAsVariant(Variant value)
```

Write-only property that sets this *Variant* to the specified *value*. If *value* is not of the same data type as specified in the *setType* property, then an attempt is made to convert it to the data type of this *Variant*. An attempt to convert data from or to and int to or from a String generates a *DataSetException*. If you

need such conversions, or prefer to perform data conversion yourself, wire the *CoerceToColumn* or *CoerceFromColumn* events as appropriate.

bigDecimal

```
public final BigDecimal getBigDecimal()  
public final void setBigDecimal(BigDecimal val)
```

Stores the value of this *Variant* as a *BigDecimal* data value.

binaryStream

```
public final InputStream getBinaryStream()  
public final void setBinaryStream(InputStream val)
```

This property was deprecated. Use the *inputStream* property.

boolean

```
public final boolean getBoolean()  
public final void setBoolean(boolean val)
```

Stores the value of this *Variant* to a **boolean** data value. Valid values are **true** or **false**.

byte

```
public final byte getByte()  
public final void setByte(byte val)
```

Stores the value of this *Variant* to a **byte** data value.

byteArray

```
public final byte[] getByteArray()
```

Retrieves the value of this *Variant* as a **byte** array.

date

```
public final java.sql.Date getDate()  
public final void setDate(java.sql.Date val)  
public final void setDate(long val)
```

Stores the value of this *Variant* as a *Date* data value. The *val* parameter can be of the *java.sql.Date* type or a **long** value.

displayValue

```
public final Object getDisplayValue()
```

Retrieves the value of the *Variant*.

double

```
public final double getDouble()
public final void setDouble(double val)
```

Stores the value of this *Variant* to a **double** data value.

float

```
public final float getFloat()
public final void setFloat(float val)
```

Stores the value of this *Variant* to a **float** data value.

InputStream

```
public final InputStream getInputStream()
public final void setInputStream(InputStream val)
```

Input streams are used to read in images such as GIF and JPEG images. They must be re-readable as they are read each time they are painted. The *InputStream.markSupported()* method must return **true** and the *InputStream.mark(0)* method should be called before the *BinaryStream* is added to the *DataSet*.

If your custom *InputStream* requires a custom editor or painter, set these *Column* level properties to your custom edit and painter classes.

val The input stream the data is read from and stored to.

int

```
public final int getInt()
public final void setInt(int val)
```

Stores the value of this *Variant* to an **int** data value.

val The new value.

long

```
public final long getLong()
public final void setLong(long val)
```

Stores the value of this *Variant* to a **long** data value.

val The new value.

null

```
public final boolean isNull()
public final void setNull(int nullType)
```

Determines whether a data value is **null**. If *null* is true, the data value is **ASSIGNED_NULL** or **UNASSIGNED_NULL**; otherwise, the data value is not **ASSIGNED_NULL** or **UNASSIGNED_NULL**.

object

```
public final Object getObject()
public final void setObject(Object val)
```

Stores the value of this *Variant* to an *Object* data value.

val The new value.

setType

```
public final int getSetType()
```

Returns the set type of the *Variant* as an integer. Possible values are the type constants of *Variant*. For example, if the data type is **boolean**, *getSetType()* returns 11, the value of the **BOOLEAN** constant. *setType* is a private variable used internally by the *Variant* class. It is used to enforce safe set operations.

short

```
public final short getShort()
public final void setShort(short val)
```

Stores the value of this *Variant* to a **short** data value.

val The new value.

string

```
public final String getString()
public final void setString(String val)
```

Stores the value of this *Variant* as a *String* data value.

val The new value as a string.

time

```
public final Time getTime()
public final void setTime(Time val)
public final void setTime(long val)
```

Stores the value of this *Variant* as a *Time* data value.

val The new value.

timestamp

```
public final Timestamp getTimestamp()
public final void setTimestamp(Timestamp val)
public final void setTimestamp(long val)
```

Stores the value of this *Variant* as a *Timestamp* data value.

val The new value.

type

```
public final int getType()
```

Returns the data type. The integer returned is the value of one of the type constants of *Variant*. For example, a data type of **double** returns a value of 7, which is the value of the DOUBLE constant.

unassignedNull

```
public final boolean isUnassignedNull()
```

Determines whether the data value is an unassigned null. An unassigned null is a value that was never assigned. If *unassignedNull* is **true**, the data was not assigned and is **null**. If it returns **false**, the data value might not be **null** or it might be an assigned **null** value.

variant

```
public final void setVariant(Variant value)
```

Sets the value of this *Variant* as a *Variant* data value.

value The new value.

Variant methods

Method	Implemented in
<code>add(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)</code>	this class
<code>clone()</code>	this class
<code>compareTo(com.borland.dx.dataset.Variant)</code>	this class
<code>currentUTCTimeMillis()</code>	this class
<code>equals(com.borland.dx.dataset.Variant)</code>	this class
<code>equals(java.lang.Object)</code>	java.lang.Object
<code>equalsInstance(com.borland.dx.dataset.Variant)</code>	this class
<code>finalize()</code>	java.lang.Object
<code>getTimeZoneOffset()</code>	this class
<code>hashCode()</code>	java.lang.Object
<code>notify()</code>	java.lang.Object
<code>notifyAll()</code>	java.lang.Object
<code>setAsObject(java.lang.Object, int)</code>	this class
<code>setAssignedNull()</code>	this class
<code>setByteArray(byte[], int)</code>	this class
<code>setFromString(int, java.lang.String)</code>	this class
<code>setTimestamp(long, int)</code>	this class
<code>setUnassignedNull()</code>	this class
<code>subtract(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)</code>	this class
<code>toString()</code>	this class
<code>typeId(java.lang.String)</code>	this class
<code>typeName(int)</code>	this class
<code>typeOf(java.lang.String)</code>	this class
<code>wait()</code>	java.lang.Object
<code>wait(long, int)</code>	java.lang.Object
<code>wait(long)</code>	java.lang.Object

`add(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)`

`public void add(Variant value2, Variant result)`

Adds a value to this *Variant*, storing the result in the *result* parameter.

value2 The value added to this *Variant*.

result The result of the two *Variant* values added together.

clone()

public Object clone()

Creates a copy of this *Variant*, returning the copied object.

Overrides `java.lang.Object.clone()`

compareTo(com.borland.dx.dataset.Variant)

public int compareTo(Variant value2)

Compares a *Variant* value to the value of this *Variant*, returning the result. If the result is zero, the two *Variants* are equal. If the returned value is less than zero (a negative integer), the value of this *Variant* is less than *value2*. If the returned value is greater than zero (a positive integer), the value of this *Variant* is greater than *value2*.

value2 The value this *Variant* is being compared to.

currentUTCTimeMillis()

public static long currentUTCTimeMillis()

Returns `System.currentTimeMillis()` minus the active time zone offset to provide the current UTC time.

equals(com.borland.dx.dataset.Variant)

public final boolean equals(Variant value)

Determines whether a *Variant* value is equal to this *Variant* value. If `equals()` returns **true**, the two *Variant* values are of the same type and are equal in value. A returned value of *false* indicates that the two values differ in value or type.

value The *Variant* value being compared to the data type and value of this *Variant*.

equalsInstance(com.borland.dx.dataset.Variant)

public boolean equalsInstance(Variant value2)

Returns **true** if the value or value instance changed. Returns **false** for *Variants* storing different object reference values that may be equal. Provides a high speed test that indicates if two variants are equal. If **true** is returned, they are equal. If **false** is returned, they might still be equal.

getTimeZoneOffset()

```
public static long getTimeZoneOffset()
```

Returns the time zone offset, in milliseconds, of the current time zone. Used internally by JDataStore.

setAsObject(java.lang.Object, int)

```
public void setAsObject(Object object, int variantType)
```

Currently used to set a *Variant* from an *Object*.

object The value to set.

variantType Variant data type that the object maps to. For example, if the object is of type Integer, then *variantType* should be Variant.INT.

setAssignedNull()

```
public final void setAssignedNull()
```

Sets the value of the *Variant* as an assigned **null**. An assigned **null** is a value that has been explicitly set to **null** in contrast to one that is simply unassigned.

setByteArray(byte[], int)

```
public final void setByteArray(byte[] val, int length)
```

Sets the value of the *Variant* to a new array of bytes.

val The new array of bytes that becomes the new value of this *Variant*.

length The length of the new byte array.

setFromString(int, java.lang.String)

```
public final void setFromString(int wantedType, String s)
```

Attempts to parse the passed string *s* to the type indicated by *wantedType*. Date values must be of the format “yyyy-mm-dd”. Time values must be of the format “hh:mm:ss”. Timestamp values must be of the format “yyyy-mm-dd hh:mm:ss.ffffff”, where *f* indicates a digit of the fractions of seconds. Boolean values are **true** for true, anything else is false.

setTimestamp(long, int)

```
public final void setTimestamp(long val, int nanos)
```

Sets the value of the *Variant* as a *Timestamp* value.

val The new value as a **long** value.

nanos The number of nanoseconds in the *Timestamp* value.

setUnassignedNull()

```
public final void setUnassignedNull()
```

Sets the value of this *Variant* to an unassigned **null**. An assigned **null** is a **null** value that has not been explicitly assigned as **null**.

subtract(com.borland.dx.dataset.Variant, com.borland.dx.dataset.Variant)

```
public void subtract(Variant value2, Variant result)
```

Subtracts a *Variant* value from the value of this *Variant*, storing the result in the *result* parameter.

value2 The value being subtracted from this *Variant*.

result The difference between the two *Variant* values.

toString()

```
public final String toString()
```

Converts the *Variant* value to a string.

Overrides `java.lang.Object.toString()`

typeid(java.lang.String)

```
public static int typeid(String name)
```

Returns the integer value that represents the specified type *name*. For example, a *name* value of `""` returns an integer of 11.

name The name of a data type. Specify the name using one of the data type constants of *Variant*. For example, `BOOLEAN` is the name of the `BOOLEAN` constant for a **boolean** data type.

typeName(int)

public static String typeName(int type)

Returns the name of a data type as a string. For example, the string representation of a BOOLEAN data type is "BOOLEAN".

type The data type. Specify the type using one of the data type constants of *Variant*. For example, BOOLEAN is the name of the constant for a **boolean** data type.

typeOf(java.lang.String)

public static int typeOf(String typeName)

Returns an integer that identifies the data type specified in the *typeName* parameter.

typeName The name of a data type as a string. For example, the string "BOOLEAN" results in an integer value of 11, which is the value of the BOOLEAN constant.

VariantException class

dx.dataset package

Extends java.lang.RuntimeException

Implements java.io.Serializable

An exception thrown when an error occurs in a *Variant* object.

VariantException constructors

VariantException(java.lang.String)

public VariantException(String error)

Constructs a *VariantException* object that contains the error that occurred in the *Variant*.

error The string containing the error that occurred.

VariantException properties

Property	Implemented in
class*	java.lang.Object
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

VariantException methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
fillInStackTrace()	java.lang.Throwable
finalize()	java.lang.Object
fire(java.lang.String)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
printStackTrace()	java.lang.Throwable
printStackTrace(java.io.PrintStream)	java.lang.Throwable
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
toString()	java.lang.Throwable
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

fire(java.lang.String)

public static final void fire(String error)

Throws a *VariantException*.

error The string containing the error that occurred.

dx.sql.dataset package

The *dx.sql.dataset* package contains classes and interfaces that provide data connectivity functionality that is JDBC specific. It collects specific provider/resolver implementations for better support of multi-tier designs and other provider/resolver implementations. Its classes are used in conjunction with those in the *com.borland.dx.dataset* package, which provides general routines for data connectivity, management, and manipulation.

The *dx.sql.dataset* package includes the following types of functionality:

Connection	Login to remote data servers through JDBC, and handling of connection events.
Providing	Obtaining data from the remote database through SQL statements or stored procedures.
Resolving	Updating of data from the local <i>DataSet</i> back to the original source of the data.

The *dx.sql.dataset* package contains the following types of classes:

- Connecting and data set classes
- Event, listener, and adapter classes
- Descriptor classes
- Exception classes
- Provider classes
- Resolver classes
- Miscellaneous dataset classes

This package includes the following BeanInfo classes:

- DatabaseBeanInfo
- ProcedureDataSetBeanInfo
- ProcedureResolverBeanInfo
- QueryDataSetBeanInfo
- QueryResolverBeanInfo

The following classes, components, and interfaces in this package are used internally by classes in this and other *com.borland* packages. These classes, components, and interfaces are not intended for general use and are not documented. Do not use them directly in your application.

- DataModelProvider
- DesignerConnectionCache
- JdbcProvider
- QueryAnalyzer
- ResolutionManager
- RuntimeMetaData
- SQLElement
- SQLToken
- Task
- TransactionSupport
- UniqueQueryAnalyzer

For more information, visit the database newsgroup. Details on newsgroups can be found at <http://www.borland.com/newsgroups>. The database newsgroup is dedicated to issues about writing database applications and is actively monitored by our support engineers as well as the Development team.

Interfaces

ConnectionUpdateListener	DefaultResolver	Load
SQLDialect	SQLToken	Task
TransactionSupport		

Classes and components

ConnectionDescriptor	ConnectionUpdateAdapter
ConnectionUpdateEvent	Database
DatabaseBeanInfo	DataModelProvider
DesignerConnectionCache	JdbcProvider
OracleProcedureProvider	ProcedureDataSet
ProcedureDataSetBeanInfo	ProcedureDescriptor
ProcedureProvider	ProcedureResolver
ProcedureResolverBeanInfo	QueryDataSet
QueryDataSetBeanInfo	QueryDescriptor
QueryProvider	QueryResolver

QueryResolverBeanInfo	ResolutionException
ResolutionManager	ResolveError
RuntimeMetaData	SQLElement
SQLResolutionManager	SQLResolver
UniqueQueryAnalyzer	

Overview of classes in the *com.borland.dx.sql.dataset* package

Connecting and data set classes

Database	Encapsulates a database connection through JDBC to a SQL server and provides lightweight transaction support using local caching. Required for accessing data on a SQL server.
ProcedureDataSet	Extends <i>StorageDataSet</i> , to run a stored procedure against a SQL database. Supports executing stored procedures via a SQL query.
QueryDataSet	Extends <i>StorageDataSet</i> , to run a query statement against a table in a SQL database. Requires a <i>Database</i> component and a <i>QueryDescriptor</i> . The query result set is stored in this component, allowing flexible navigation of the results.

Event, listener, and adapter classes

ConnectionUpdateEvent	Used when a database connection is about to close, or is closed. Also used when the transaction <i>isolationLevel</i> has changed.
ConnectionUpdateListener	A listener interface for <i>ConnectionUpdateEvent</i> .
ConnectionUpdateAdapter	An adapter class for <i>ConnectionUpdateListener</i> .

Descriptor classes

ConnectionDescriptor	Stores properties related to connecting to a SQL database, such as the connection URL, user name, and password. Required for accessing data on a SQL server.
ProcedureDescriptor	Stores properties of a stored procedure (<i>ProcedureDataSet</i>) against a SQL database.
QueryDescriptor	Stores properties that set a query statement to run against a SQL database. Required for accessing SQL table data.

Exception classes

ResolutionException	Used whenever there is an error during resolution of a <i>DataSet</i> . Extends <i>DataSetException</i> .
---------------------	---

Provider classes

Load	Constants that determine how data is loaded into a <i>DataSet</i> .
OracleProcedureProvider	A provider class for Oracle stored procedures.
ProcedureProvider	A provider component for stored procedures.
QueryProvider	A provider class for queries executed against a SQL database.

Resolver classes

DefaultResolver	An interface for resolver functionality.
ProcedureResolver	A class implementing <i>DefaultResolver</i> specifically for resolving data changes using a stored procedure.
QueryResolver	Used to customize <i>DataSet</i> resolving events and properties.
SQLResolver	Allows for alternate implementations of the behavior required to save changes made to a <i>DataSet</i> to its database data source.
SQLResolutionManager	Manages the resolution of one or more <i>DataSets</i> to a <i>Database</i> component.

Miscellaneous dataset classes

SQLDialect

Defines constants for SQL database servers.

ConnectionDescriptor class

dx.sql.dataset package

Extends java.lang.Object**Implements** java.io.Serializable, java.lang.Cloneable

The *ConnectionDescriptor* object stores properties related to connecting to a SQL database. Its main properties are:

- *connectionURL* (the Universal Resource Locator of the database)
- *userName*
- *password*
- *driver*
- (extended driver) *properties*

Both the *ConnectionDescriptor* object and the *Database* component are required elements when accessing data that is stored on a SQL server.

The information stored in the *ConnectionDescriptor* can be accessed through the user interface by inspecting the *connection* property of a *Database* object. To work with this object programmatically, you set its properties when instantiating the *ConnectionDescriptor*, or by its write accessors.

For application design issues when connecting to a database (including displaying the username/password dialog) see the “About” section for the *Database* component.

ConnectionDescriptor constructors

ConnectionDescriptor(com.borland.dx.sql.dataset.ConnectionDescriptor)

```
public ConnectionDescriptor(ConnectionDescriptor cDesc)
```

Constructs a *ConnectionDescriptor* using the property values from the *ConnectionDescriptor* object specified as *cDesc*.

ConnectionDescriptor(java.lang.String)

```
public ConnectionDescriptor(String connectionURL)
```

Constructs a *ConnectionDescriptor* with the specified URL to the database.

ConnectionDescriptor(java.lang.String, java.lang.String)

```
public ConnectionDescriptor(String connectionURL, String userName)
```

Constructs a *ConnectionDescriptor* with the specified connection URL to the database and user name.

ConnectionDescriptor(java.lang.String, java.lang.String, java.lang.String)

```
public ConnectionDescriptor(String connectionURL, String userName, String password)
```

Constructs a *ConnectionDescriptor* with the specified connection URL to the database, user name, and password.

ConnectionDescriptor(java.lang.String, java.lang.String, java.lang.String, boolean)

```
public ConnectionDescriptor(String connectionURL, String userName, String password, boolean
    promptPassword)
```

Constructs a *ConnectionDescriptor* with the specified connection URL to the database, user name, password, and whether to prompt for the password each time or store the password in the *ConnectionDescriptor*.

ConnectionDescriptor(java.lang.String, java.lang.String, java.lang.String, boolean, java.lang.String)

```
public ConnectionDescriptor(String connectionURL, String userName, String password, boolean
    promptPassword, String driver)
```

Constructs a *ConnectionDescriptor* with the specified connection URL to the database, user name, password, whether to prompt for the password each time or store the password in the *ConnectionDescriptor*, and the driver class to use when connecting to the *Database*.

ConnectionDescriptor(java.lang.String, java.lang.String, java.lang.String, boolean, java.lang.String, java.util.Properties)

```
public ConnectionDescriptor(String connectionURL, String userName, String password, boolean
    promptPassword, String driver, Properties properties)
```

Constructs a *ConnectionDescriptor* with the specified connection URL to the database, user name, password, whether to prompt for the password each time or store the password in the *ConnectionDescriptor*, the driver class to use when connecting to the *Database*, and the instance of a *java.util.Properties* that stores extended driver properties to use when connecting.

Not all drivers support connecting to a database using a *java.util.Properties* object. Check your driver documentation for more information on whether it supports this feature or not.

ConnectionDescriptor properties

Property	Implemented in
class*	java.lang.Object
complete*	this class
connectionURL	this class
driver	this class
password	this class
promptPassword	this class
properties	this class
userName	this class

complete

public synchronized boolean isComplete()

Read-only property that returns **true** if all the *ConnectionDescriptor* parameters have been set.

connectionURL

public synchronized String getConnectionURL()

public synchronized void setConnectionURL(String url)

Stores the name of the connection Universal Resource Locator (URL) for the database. The format consists of the URL type ("jdbc" for JDBC database access), followed by driver-specific information, separated by colons. The driver-specific information is driver/server dependent. It typically includes the driver name followed by the data source name. Refer to the documentation for your driver for more information on this property.

When you successfully connect to a database, the database URL is saved in the `jbuilder.properties` file. The URL history is read from the properties file when you click the Choose URL button from the *Connection* custom property editor.

driver

public synchronized String getDriver()

public synchronized void setDriver(String driver)

The driver class to use when connecting to the *Database*, for example, `jdbc.odbc.jdbcodbcDriver`. This property is optional if you have already registered the driver class with JDBC—if you specify it, it will not re-register a previously registered class. If the driver class has not been registered with JDBC, specify it here to have `JDataStore` register the driver class with JDBC,

and have JDBC load the driver in memory before attempting the connection to the *Database*.

password

```
public synchronized String getPassword()
public synchronized void setPassword(String password)
```

Stores the password used to connect to the database in the source code as unencrypted text. For maximum security, do not specify the password using this property but set the *promptPassword* property to **true**.

When accessing this property through the user-interface, an asterisk (*) appears for each character that you type in this field.

promptPassword

```
public synchronized boolean isPromptPassword()
public synchronized void setPromptPassword(boolean prompt)
```

Stores whether the password is maintained in the Java code or not, and therefore, whether the user should be prompted for a password each time a connection to the database is made. The default for this method is **false**.

When prompted for the password, you can optionally specify an alternate user name for the connection which overrides the *userName* property setting at runtime.

For application design issues when connecting to a database and setting this property to true, see the About section for the *Database* component.

properties

```
public synchronized Properties getProperties()
public synchronized void setProperties(Properties properties)
```

Some drivers may require or optionally accept additional connection parameters when connecting to a database. Such drivers take a *java.util.Properties* object that contains the extended properties. Use this property to specify the *Properties* object.

Note Not all drivers support connecting to a database using a *java.util.Properties* object. Check your driver documentation for more information on whether it supports this feature or not.

userName

```
public synchronized String getUserName()
public synchronized void setUserName(String userName)
```

Stores the user name used to connect to the database.

ConnectionDescriptor methods

Method	Implemented in
arrayToProperties(java.lang.String[][])	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

arrayToProperties(java.lang.String[][])

public static Properties arrayToProperties(String[][] array)

A service method to convert a 2D string array into a *Properties* object. Used by the *ConnectionDescriptor* editor in generating code. Can also be used by developers. This method returns a *Properties* object containing key/value pairs.

array A two-dimensional array of strings corresponding to key/value pairs.

toString()

public String toString()

Returns a *String* equivalent of the property values stored in the *ConnectionDescriptor*. The password is not included in the resulting *String*.

Overrides java.lang.Object.toString()

ConnectionUpdateAdapter class

dx.sql.dataset package

Extends java.lang.Object

Implements com.borland.dx.sql.dataset.ConnectionUpdateListener,
java.util.EventListener

This is an adapter class for *ConnectionUpdateListener*, which provides notification before and after closing a database connection or changing the attributes of the JDBC connection.

ConnectionUpdateAdapter properties

Property	Implemented in
class*	java.lang.Object

ConnectionUpdateAdapter methods

Method	Implemented in
canChangeConnection (com.borland.dx.sql.dataset.ConnectionUpdateEvent)	this class
clone()	java.lang.Object
connectionChanged (com.borland.dx.sql.dataset.ConnectionUpdateEvent)	this class
connectionClosed (com.borland.dx.sql.dataset.ConnectionUpdateEvent)	this class
connectionOpening (com.borland.dx.sql.dataset.ConnectionUpdateEvent)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ConnectionUpdateEvent class

dx.sql.dataset package

Extends java.util.EventObject

Implements com.borland.jb.util.ExceptionDispatch, java.io.Serializable

This class is used to inform the *ConnectionUpdateListener* before and after closing a database connection or when changing the attributes of the JDBC connection.

ConnectionUpdateEvent variables

Variable	Defined in
CAN_CLOSE	this class
CHANGED	this class
CLOSED	this class
OPENING	this class
source	java.util.EventObject

CAN_CLOSE

public static final int CAN_CLOSE = 3

Asking to close a database connection.

CHANGED

public static final int CHANGED = 1

Changing a database connection.

CLOSED

public static final int CLOSED = 2

Closing a database connection.

OPENING

public static final int OPENING = 4

Opening a database connection.

ConnectionUpdateEvent constructors

ConnectionUpdateEvent(java.lang.Object)

public ConnectionUpdateEvent(Object source)

Constructs a *ConnectionUpdateEvent* object.

source The class that issued the event.

ConnectionUpdateEvent properties

Property	Implemented in
class*	java.lang.Object
source*	java.util.EventObject

ConnectionUpdateEvent methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
exceptionDispatch(java.util.EventListener)	this class
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.util.EventObject
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ConnectionUpdateListener interface

dx.sql.dataset package

Extends java.util.EventListener

Implemented by com.borland.dx.sql.dataset.ConnectionUpdateAdapter,
com.borland.dx.sql.dataset.JdbcProvider,
com.borland.dx.sql.dataset.OracleProcedureProvider,
com.borland.dx.sql.dataset.ProcedureProvider,
com.borland.dx.sql.dataset.QueryProvider

This interface is used for notification before and after closing a database connection or changing the attributes of the JDBC connection.

ConnectionUpdateListener methods

Method	Implemented in
<code>canChangeConnection</code> (<code>com.borland.dx.sql.dataset.ConnectionUpdateEvent</code>)	this class
<code>connectionChanged</code> (<code>com.borland.dx.sql.dataset.ConnectionUpdateEvent</code>)	this class
<code>connectionClosed</code> (<code>com.borland.dx.sql.dataset.ConnectionUpdateEvent</code>)	this class
<code>connectionOpening</code> (<code>com.borland.dx.sql.dataset.ConnectionUpdateEvent</code>)	this class

`canChangeConnection(com.borland.dx.sql.dataset.ConnectionUpdateEvent)`

`public void canChangeConnection(ConnectionUpdateEvent event)`

The event that gets fired when a database connection is about to change. Throwing an *Exception* will block the connection change.

`connectionChanged(com.borland.dx.sql.dataset.ConnectionUpdateEvent)`

`public void connectionChanged(ConnectionUpdateEvent event)`

The event that gets fired when a database connection has changed.

`connectionClosed(com.borland.dx.sql.dataset.ConnectionUpdateEvent)`

`public void connectionClosed(ConnectionUpdateEvent event)`

The event that gets fired when a database connection is closed.

`connectionOpening(com.borland.dx.sql.dataset.ConnectionUpdateEvent)`

`public void connectionOpening(ConnectionUpdateEvent event)`

The event that gets fired when a database connection is being opened.

Database component

`dx.sql.dataset` package

Extends `java.lang.Object`

Implements `com.borland.dx.dataset.Designable`, `java.io.Serializable`

The *Database* component is a required element of any application accessing data stored on a SQL server. It encapsulates a database connection through JDBC to the SQL server and also provides lightweight transaction support.

When used with a *QueryDataSet* or *ProcedureDataSet* component, data is retrieved from the external database into a local cache (*DataSet*) on the user's system. All row data for a particular *DataSet* is cached on the user's system as a single unit. Changes made to the local copy of the data in the *DataSet* are internally recorded as deletes, inserts and updates. When all changes to the *DataSet* are complete, you then save the changes back to the original database by calling the *saveChanges()* method with one or more *DataSet* components.

You can connect several *QueryDataSet* or *ProcedureDataSet* components to a *Database* component, however, some SQL servers allow only one active query at a time on a connection. Check your server documentation to see if this is applicable to the SQL server you are accessing.

Setting connection properties

The *Database* component has an associated *connection* property that stores the connection properties of *userName*, *password*, and *connectionUrl* of the database. These properties are stored in the *ConnectionDescriptor* object. When the necessary properties for the connection have been supplied, the connection can be opened explicitly or automatically. When explicitly connecting, use the *openConnection()* method. The connection is opened automatically when you explicitly open the *DataSet*, or when a UI control requests data that is obtained through the database connection.

If all needed properties have been set, a connection is attempted when any of the following situations occur:

- an explicit *openConnection()* method call is made
- a *QueryDataSet* that relies on data from this connection requests it

When attempting to make the connection to the *Database*, the appropriate driver needed to access the remote server is loaded. If the remote server driver information is not available in the system registry, you can specify the driver using the *addDriver()* method, in the *ConnectionDescriptor* object, or in the Connection property editor.

Application design

It is strongly recommended that you include all DataExpress components (database connections, queries, *DataStores*, and so on) in a *DataModule*. The *DataModule* is a specialized container for data access components and their properties. Consolidation of these components in a single container clarifies an application's design and increases the reusability of the data access components.

The isolation level, specified by the *transactionIsolation* property, is used when saving data changes back to the external database table.

When you need special transaction logic, use the *saveChanges(com.borland.dx.dataset.DataSet[], boolean, boolean, boolean)* method.

By setting its final parameter *resetPendingStatus* to false, this method offers the flexibility of not resetting the pending resolved status bits through the call to the *saveChanges* method. When you want to reset the pending resolved status bit, call the *resetPendingStatus* method. This allows you, for example, to save changes made to several *DataSets* in a single transaction, and to rollback all changes while still retaining all the changed data in both *DataSets*.

When designing your application that involves prompting for a password, set the *promptPassword* property to true, then call the *openConnection()* method for your database when you want the username/password dialog to appear. If you want the username/password dialog to appear as soon as your application loads, call the *openConnection()* method at the end of the main frame's *jbInit()* method.

If the user cancels the password dialog, your application can detect a *DataSetException* of type *CONNECTION_DESCRIPTOR_NOT_SET* and take the appropriate action. The application could either terminate or disable data-access functions.

When you no longer need a *Database* connection, you should explicitly call the *Database.closeConnection()* method in your application. This ensures that *Database* classes which hold references to JDBC connections are automatically closed when the *Database* object is garbage collected.

Debugging JDBC connections

When debugging JDBC connection issues, you can add the following line of code to your application before the line of code that executes your query or stored procedure:

```
java.sql.DriverManager.setLogStream(System.out);
```

This generates (verbose) output from the JDBC driver that shows what is happening and in what sequence. Examining this output may help determine the source of JDBC connection related problems in your application.

To turn off the debugging output, use the following code:

```
java.sql.DriverManager.setLogStream(null);
```

If you're connecting to data using ODBC drivers under the MicroSoft Windows operating system, enable ODBC logging through the Control Panel program. The calls that take place are displayed, enabling you to track what is being sent to the ODBC driver.

Database variables

Variable	Defined in
DEFAULT_DRIVERS	this class

DEFAULT_DRIVERS

```
public static String DEFAULT_DRIVERS = "sun.jdbc.odbc.JdbcOdbcDriver"
```

The default driver to use; the JDBC-ODBC Bridge driver from JavaSoft.

Database constructors

Database()

```
public Database()
```

Creates a *Database* object and sets the transaction isolation level to *ReadCommitted*.

Database properties

Property	Implemented in
autoCommit	this class
class*	java.lang.Object
connection	this class
identifierQuoteChar	this class
jdbcConnection	this class
maxStatements*	this class
metaData*	this class
open*	this class
runtimeMetaData	this class
SQLDialect	this class
transactionIsolation	this class
useCaseSensitiveId	this class
useCaseSensitiveQuotedId	this class
useSchemaName	this class
useSetObjectForStreams	this class
useSetObjectForStrings	this class
useSpacePadding	this class
useStatementCaching	this class
useTableName	this class
useTransactions	this class

autoCommit

```
public final boolean getAutoCommit()
public final void setAutoCommit(boolean enable)
```

Specifies whether *autoCommit* is enabled (**true**) or not (**false**). If **true**, each SQL statement is executed and implicitly committed as an individual transaction. If **false**, all SQL statements are executed in a single transaction that is explicitly terminated by a *commit()* or *rollback()*. Where a SQL statement returns a result set that is stored in a *QueryDataSet* or a *ProcedureDataSet*, the statement completes when the last row of the result set has been retrieved.

This property defaults to **true**. On failure, it throws a *DataSetException* or a *SQLException* as applicable.

connection

```
public ConnectionDescriptor getConnection()
public void setConnection(ConnectionDescriptor connectionDescriptor)
```

The *connection* property is a complex property, containing all the information needed by the *Database* to establish a JDBC connection (including a Universal Resource Locator, a user name, and a password). The values are stored in a *ConnectionDescriptor* object.

You can read from or write to this property at any time, however, the new *ConnectionDescriptor* will not be applied until the next time the *Database* is explicitly opened.

identifierQuoteChar

```
public char getIdentifierQuoteChar()
public void setIdentifierQuoteChar(char quoteChar)
```

Specifies the server's quote character. Setting this property to `'\0'` instructs *JDataStore* to not use quoted identifiers in generated queries.

jdbcConnection

```
public final java.sql.Connection getJdbcConnection()
public final void setJdbcConnection(Connection connection)
```

Specifies the connection object to use. Use the setter method when a JDBC connection has already been established and you want *DataExpress* components to share that connection. To use an explicit *Connection* object when connecting to a *Database*:

- 1 create a *java.sql.Connection* object
- 2 set this property to the *Connection* object
- 3 instantiate the *Database*

Note Be aware of any issues that sharing a *Connection* can have. If the connection is closed while the *DataSet* is still using it, unknown behavior can result. Additionally, there may be issues with servers that only allow one active query per *Connection*.

Use the getter of this property to obtain the JDBC *Connection* object.

maxStatements

```
public final int getMaxStatements()
```

Read-only property that returns the maximum number of statements that the server supports.

metaData

```
public final synchronized DatabaseMetaData getMetaData()
```

Read-only property that returns the metadata for the *Database* object. Metadata is that information that describes the database, for example, a listing of column names and data types. On failure, this method throws a *SQLException*.

open

```
public boolean isOpen()
```

Read-only property that returns whether the database connection is open.

runtimeMetaData

```
public final synchronized RuntimeMetaData getRuntimeMetaData()
public final synchronized void setRuntimeMetaData(RuntimeMetaData runtimeMeta)
```

This property is used internally by other *com.borland* classes. You should never use this property directly.

SQLDialect

```
public final int getSQLDialect()
public final void setSQLDialect(int dialect)
```

Specifies the SQL dialect that your server is based on. This is not required, but may be useful. Valid values for *dialect* are defined in *SQLDialect*.

transactionIsolation

```
public final int getTransactionIsolation()
public final synchronized void setTransactionIsolation(int level)
```

Specifies the transaction isolation level for the *Database* object. A single transaction is used when making changes to the external database table. On

failure, this method throws a *SQLException*. Accepted values for the transaction isolation are listed under *java.sql.connection*.

The *setTransactionIsolation()* method checks to see if the driver supports transactions and if so, if the specified isolation level is set. If it isn't, a higher level of isolation (more restrictive) is attempted according to the following table:

Level failed	Next level attempted
TRANSACTION_NONE	none
TRANSACTION_READ_UNCOMMITTED	TRANSACTION_READ_COMMITTED
TRANSACTION_READ_COMMITTED	TRANSACTION_REPEATABLE_READ
TRANSACTION_REPEATABLE_READ	TRANSACTION_SERIALIZABLE
TRANSACTION_SERIALIZABLE	<i>DataSetException</i> thrown

If the connection is open, this property may be set at a higher level than you selected. If a supported isolation level is not found, a *DataSetException* of *TransactionIsolationLevelNotSupported* is thrown. This method also throws a *SQLException* as appropriate.

You can change the transaction isolation level only on a newly opened connection—attempting to do so in the middle of a transaction will generate a *SQLException*.

See also *java.sql.DatabaseMetaData#supportsTransactionIsolationLevel*

useCaseSensitiveId

```
public boolean isUseCaseSensitiveId()
public void setUseCaseSensitiveId(boolean caseSensitive)
```

Controls whether an identifier in an SQL string that doesn't have quotes is treated as case sensitive by the database.

If this property is not set, it defaults to the metadata value reported by the JDBC driver.

useCaseSensitiveQuotedId

```
public boolean isUseCaseSensitiveQuotedId()
public void setUseCaseSensitiveQuotedId(boolean caseSensitive)
```

Controls whether a quoted identifier in an SQL string is treated as case sensitive by the database.

If this property is not set, it defaults to the metadata value reported by the JDBC driver.

useSchemaName

```
public boolean isUseSchemaName()
public void setUseSchemaName(boolean useSchemaName)
```

Determines whether the user name (that was used to connect to the *Database*) should be included with all metadata discovery requests sent to the server and defaults to **false**. This property does not apply to queries executed against the server.

Note This property is required for some servers; check your server software documentation to determine whether your server software requires an explicit user name for metadata discover requests.

useSetObjectForStreams

```
public boolean isUseSetObjectForStreams()
public void setUseSetObjectForStreams(boolean useSetObjectForStreams)
```

Controls which method is used to save a *BinaryStream* value when resolving data. If **true**, the *setObject()* method is used. If **false**, *setBinaryStream()* is used on the *PreparedStatement*. This property defaults to **false** for the ODBC bridge and **true** for all other drivers.

useSetObjectForStrings

```
public boolean isUseSetObjectForStrings()
public void setUseSetObjectForStrings(boolean useSetObjectForStrings)
```

Controls which method is used to save a *String* value when resolving data. If **true**, the *setObject()* method is used. If **false**, *setString()* is used on the *PreparedStatement*. This property defaults to **true**.

useSpacePadding

```
public boolean isUseSpacePadding()
public void setUseSpacePadding(boolean useSpacePadding)
```

Controls if a *CHAR* field should be space padded or not. This can sometimes help to work around certain database driver bugs.

useStatementCaching

```
public boolean isUseStatementCaching()
public void setUseStatementCaching(boolean useStatementCaching)
```

Controls whether the JDBC statements should be reused.

useTableName

```
public boolean isUseTableName()
public void setUseTableName(boolean useTableName)
```

Determines whether the table name should be prepended to all field names in all queries executed against SQL server data. This property defaults to **false**.

Note This property is required for some servers; check your server software documentation to determine whether your server software requires an explicit table name for query execution. This property can be helpful for drivers that have difficulty in parsing quoted field names.

useTransactions

```
public boolean isUseTransactions()
public void setUseTransactions(boolean useTransactions)
```

Controls if *saveChanges* should use transactions. By default, it will use transactions if the associated JDBC driver returns **true** for its implementation of *DatabaseMetaData.supportsTransactions()*.

Database methods

Method	Implemented in
addDriver(java.lang.String, boolean)	this class
addDriver(java.lang.String)	this class
addDrivers(java.lang.String)	this class
clone()	java.lang.Object
closeConnection()	this class
commit()	this class
createCallableStatement(java.lang.String)	this class
createPreparedStatement(java.lang.String)	this class
createStatement()	this class
equals(java.lang.Object)	java.lang.Object
executeStatement(java.lang.String)	this class
finalize()	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
openConnection()	this class
resetPendingStatus(com.borland.dx.dataset.DataSet[], boolean)	this class
resultSetToDataSet(com.borland.dx.dataset.StorageDataSet, java.sql.ResultSet)	this class
resultSetToDataSet(java.sql.ResultSet)	this class

Method	Implemented in
rollback()	this class
saveChanges(com.borland.dx.dataset.DataSet)	this class
saveChanges(com.borland.dx.dataset.DataSet[], boolean, boolean, boolean)	this class
saveChanges(com.borland.dx.dataset.DataSet[], boolean)	this class
saveChanges(com.borland.dx.dataset.DataSet[])	this class
storesLowerCaseIdentifiers()	this class
storesUpperCaseIdentifiers()	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

addDriver(java.lang.String)

public final void addDriver(String driver)

Adds the specified *driver* name to the jdbc.drivers property in the system property list (if it hasn't already been registered) and instructs JDBC to load the driver prior to attempting the database connection.

Currently, note that the JDBC-ODBC driver is always added to the property setting.

driver The class name of the driver to register.

addDriver(java.lang.String, boolean)

public static void addDriver(String driver, boolean multiple)

Adds the specified *driver* name to the jdbc.drivers property in the system property list (if it hasn't already been registered) and instructs JDBC to load the driver prior to attempting the database connection.

Currently, note that the JDBC-ODBC driver is always added to the property setting.

driver The class name of the driver to register. Multiple drivers can be specified by separating each driver with a semi-colon (;) character.

multiple Specifies if multiple drivers are used.

addDrivers(java.lang.String)

```
public static void addDrivers(String driver)
```

Adds the specified *driver* name(s) to the jdbc.drivers property in the system property list (if they haven't already been registered) and instructs JDBC to load the driver(s) prior to attempting the database connection.

Currently, note that the JDBC-ODBC driver is always added to the property setting.

driver

The class name(s) of the driver to register. Multiple drivers can be specified by separating each driver with a semi-colon (;) character.

closeConnection()

```
public final synchronized void closeConnection()
```

Closes an active database connection by setting the JDBC *Connection* object's transaction mode to *AutoCommit*, invoking the *Connection* object's *close()* method, then resetting the *Connection* object to **null**. On failure, this method throws a *SQLException*.

You should explicitly call this method in your application when you no longer need a *Database* connection to assure that all connection objects (*Database*, *Connection* and so on) are properly released when garbage collection occurs.

commit()

```
public void commit()
```

Commits changes back to the server.

createCallableStatement(java.lang.String)

```
public final synchronized CallableStatement createCallableStatement(String query)
```

Opens a database connection, parses a SQL stored procedure query string that can contain zero or more "?" parameters, prepares, and returns it in a *CallableStatement*. A SQL statement with IN and/or OUT parameters can be pre-compiled and stored in a *CallableStatement* object.

Check the documentation for your driver software to see if callable statements are supported by your driver.

createPreparedStatement(java.lang.String)

```
public final synchronized PreparedStatement createPreparedStatement(String query)
```

Opens a database connection, parses a query string, prepares it, and returns it in a *PreparedStatement* object. This *PreparedStatement* object can then be used

to efficiently execute the statement, instead of the *Statement* object which must be compiled at each execution. A *PreparedStatement* object is used in situations where the exact same query is executed frequently. A SQL statement with or without IN parameters can be pre-compiled and stored in a *PreparedStatement* object.

To the end user of your application, there is no difference between using a pre-compiled *PreparedStatement* object or a non-compiled equivalent. The developer however, may find slight differences when exception events are thrown. Precompiling a query into a *PreparedStatement* can offer improved performance however not all queries can be precompiled and not all drivers support precompilation. Check the documentation for your driver software to see if precompilation is possible and the conditions under which it is possible.

On failure, this method throws a *SQLException* or *DataSetException* as appropriate.

createStatement()

```
public final synchronized Statement createStatement()
```

Opens the *Connection* to the database and returns a *Statement* object. The *Statement* object returned can be used for executing a static query (a query without parameters). On failure, this method throws a *SQLException*.

executeStatement(java.lang.String)

```
public int executeStatement(String statementString)
```

Executes the SQL statement specified by the *statementString* parameter. Use this method to execute arbitrary SQL statements, including DDL, that do not yield a *ResultSet*.

This method does not execute parameterized SQL statements. Equivalent methods that permit parameter passing are available as static methods of the *QueryProvider* (*executeStatement()*) and *ProcedureProvider* (*callProcedure()*) components. For more information on which of these methods to use, see “Executing queries” in the About section of the *QueryDescriptor* class.

For example,

```
executeStatement("create table my_table ( name varchar(20) );");
```

finalize()

```
protected void finalize()
```

Closes the connection to the database server and releases all objects for garbage collection.

Overrides `java.lang.Object.finalize()`

openConnection()

```
public final synchronized void openConnection()
```

Connects to a driver using the *ConnectionDescriptor* object property values of *userName*, *password* and *connectionURL*. Each registered driver is loaded until one is found that can process the specified URL. The transaction isolation level is set and *DatabaseMetaData* object is obtained. The *DatabaseMetaData* object contains information about the Database, for example, the list of column names, data types and views in a database.

On failure, this method throws a *DataSetException*.

resetPendingStatus(com.borland.dx.dataset.DataSet[], boolean)

```
public void resetPendingStatus(DataSet[] dataSets, boolean markResolved)
```

Resets the pending status bits. The *saveChanges(com.borland.dx.dataset.DataSet[], boolean, boolean, boolean)* method allows you to call *saveChanges* without resetting the pending resolved status bits. In such cases, use the *resetPendingStatus* method to reset the pending resolved status bits when appropriate. This feature is useful when you want special transaction logic, for example, when changes made to several *DataSets* should be saved in a single transaction. This method allows you to rollback all the changes and still have all the updates in the *DataSets*.

<i>dataSets</i>	The array of <i>DataSet</i> objects to reset status bits of.
<i>markResolved</i>	If true , status bits are reset after changes are saved (the default). If false , status bits are left in a pending state.

resultSetToDataSet(com.borland.dx.dataset.StorageDataSet, java.sql.ResultSet)

```
public StorageDataSet resultSetToDataSet(StorageDataSet dataSet, ResultSet result)
```

The data from the result set of a JDBC query or prepared statement is added to the specified *DataSet*. The *DataSet* must be closed before calling this method. This method may add columns to the *DataSet*. On error, this method throws a *DataSetException*.

<i>dataSet</i>	The <i>StorageDataSet</i> that the <i>ResultSet</i> data will be added to.
<i>result</i>	The JDBC <i>ResultSet</i> .

resultSetToDataSet(java.sql.ResultSet)

```
public StorageDataSet resultSetToDataSet(ResultSet result)
```

Creates and returns a *StorageDataSet* object that contains data from the result set of a JDBC query or prepared statement. On failure, this method throws a *SQLException* or a *DataSetException* as appropriate.

result The JDBC *ResultSet*.

rollback()

```
public void rollback()
```

Rolls back changes made to data since the last *commit()* or *rollback()* operation.

saveChanges(com.borland.dx.dataset.DataSet)

```
public final void saveChanges(DataSet dataSet)
```

Saves changes made to the local copy of the data specified by *DataSet* back to the data source. Changes to the local data are done through *DataSet* methods, either programmatically or using a data-aware control.

dataSet The *DataSet* that contains the modified data.

saveChanges(com.borland.dx.dataset.DataSet[])

```
public final void saveChanges(DataSet[] dataSets)
```

Saves changes made to data contained in the *DataSet* components listed in the array. Changes to the local data are done programmatically or using a data-aware control.

dataSets An array of *DataSet* components that contain modified data to save back to the data source.

saveChanges(com.borland.dx.dataset.DataSet[], boolean)

```
public final void saveChanges(DataSet[] dataSets, boolean doTransactions)
```

Saves changes made to one or more *DataSet* objects to the database data source.

dataSets An array of *DataSet* components that contain modified data to save back to the data source.

doTransactions Determines whether any transactions are used while resolving (**true**) or whether the user must explicitly commit or rollback changes (**false**).

saveChanges(com.borland.dx.dataset.DataSet[], boolean, boolean, boolean)

```
public final void saveChanges(DataSet[] dataSets, boolean doTransactions, boolean postEdits,
    boolean resetPendingStatus)
```

Saves changes made to one or more *DataSet* changes to the database.

<i>dataSets</i>	Array of <i>DataSet</i> components to save changes for.
<i>doTransactions</i>	If true , all changes will be in a single transaction (the default). If false , no transactions calls will be made.
<i>postEdits</i>	If true , all edits are posted before changes are saved (the default). If false , edits are not automatically saved.
<i>resetPendingStatus</i>	If true , status bits are reset after changes are saved (the default). If false , status bits are left in pending state. This allows you to reset the pending resolved bits outside the call to the <i>saveChanges</i> method. For more information see the <i>resetPendingStatus()</i> method.

storesLowerCaselIdentifiers()

```
public boolean storesLowerCaselIdentifiers()
```

Returns whether the database stores identifiers in lowercase (**true**) or not (**false**).

storesUpperCaselIdentifiers()

```
public boolean storesUpperCaselIdentifiers()
```

Returns whether the database stores identifiers in uppcase (**true**) or not (**false**).

Database event listeners

This component is a source for the following event sets.

connectionUpdate

```
public final void addConnectionUpdateListener(ConnectionUpdateListener listener)
public final void removeConnectionUpdateListener(ConnectionUpdateListener listener)
```

DefaultResolver interface

dx.sql.dataset package

Implemented by com.borland.dx.sql.dataset.ITSResolutionManager,
com.borland.dx.sql.dataset.SQLResolutionManager

The *DefaultResolver* interface collects behavior for supplying a *Resolver* object to the *ResolutionManager*. Objects implementing this interface are responsible for specifying an initialized *Resolver* object to the *ResolutionManager*.

Whenever the *ResolutionManager* needs a *Resolver* object, it invokes the *getResolver()* method and passes to it the current *DataSet* being resolved. An implementation of this object can either return an instance of a specific type of *Resolver*, or can extract the *resolver* property (if one exists) from the *DataSet* passed in. If no *Resolver* property is set for the *DataSet*, it is this object's responsibility to return an instance to a *Resolver* for the *ResolutionManager* to use.

The *Database* component implements this class and uses the *QueryResolver* as its default *Resolver* object.

DefaultResolver methods

Method	Implemented in
getResolver(com.borland.dx.dataset.DataSet)	this class

getResolver(com.borland.dx.dataset.DataSet)

public Resolver getResolver(DataSet dataSet)

Returns the *Resolver* associated with the specified *dataSet*. Implementations of this method are responsible for doing any initialization required by the *Resolver* object.

Load interface

dx.sql.dataset package

The *Load* interface defines constants used to control how data is loaded into a *DataSet*. Values set in these editors are stored in the *QueryDescriptor* or *ProcedureDescriptor* object, as appropriate. Programmatically, set the *loadOption* property for the descriptor class for the *QueryDataSet* or *ProcedureDataSet*.

If any load option other than *ALL* is used, when *Database.saveChanges()* is called, the query is terminated and the JDBC *ResultSet* released. The only

way to get additional data is to refetch the data. See the descriptions of each constant for additional information specific to that load option.

Load variables

Variable	Defined in
ALL	this class
AS_NEEDED	this class
ASYNCHRONOUS	this class
UNCACHED	this class

ALL

```
public static final int ALL = 0
```

Load all data in a single fetch. The *JDBC ResultSet* is closed after use. Note that if the (*StorageDataSet's*) *maxRows* property is set, then only that number of rows are loaded; no other rows in the *ResultSet* will ever be loaded into the *DataSet*.

AS_NEEDED

```
public static final int AS_NEEDED = 2
```

An initial number of rows is loaded. Then, whenever a navigation beyond the last loaded row is attempted, another set of rows are loaded. Enlarging a *JdbTable* or reducing the height of its rows so that there is additional space to display data does not cause additional rows to be loaded nor will moving to the last row of the grid using the scrollbar or the *PgUp* or *PgDown* keys.

The following actions cause an additional set of rows to be added:

- *DataSet.next()* when positioned on the last row
- *DataSet.last()*
- *DataSet.goToRow(int)* past the last row

The number of rows loaded at a time is controlled by the (*StorageDataSet's*) *maxRows* property at runtime and *maxDesignRows* property in the UI Designer. If the *maxRows* property is not set (its default value is -1) when using this constant, 25 rows are loaded at a time.

When using this constant, a call to *getRowCount()* returns the number of rows loaded; it doesn't return the number of rows in the *ResultSet*. Similarly, locates perform the search on loaded rows only.

The *JDBC ResultSet* is kept open until all the data is loaded. There is no notification that there are additional rows to be loaded.

ASYNCHRONOUS

```
public static final int ASYNCHRONOUS = 1
```

A new thread is created for executing the query and fetching the results. This can yield better performance. However, running the query in a separate thread could introduce the possibility of a dead-lock.

Note Running a query asynchronously may cause some methods to fail if all of the rows have not been retrieved. For example, your code might ask for rows that are not yet retrieved by the thread executing the query and therefore display an error about an empty *DataSet*, a call to *getRowCount()* could return 0, and so on. To determine if a query has completed executing, call the *LoadListener.dataLoaded(com.borland.dx.dataset.LoadEvent)* method before executing your code that depends on the data being there. You can also call the *StorageDataSet.closeProvider(true)* method to cause JBuilder to fetch any remaining rows.

UNCACHED

```
public static final int UNCACHED = 4
```

Initially, one row is loaded. Whenever a navigation beyond the loaded row is attempted, another row is loaded that replaces the previously loaded row. The *DataSet* will keep changes as normal. The JDBC *ResultSet* is kept open until the last record is read.

The description for *AS_NEEDED* applies to this constant as well, except that the number of rows loaded in this case, is one.

OracleProcedureProvider class

dx.sql.dataset package

Extends com.borland.dx.sql.dataset.ProcedureProvider

Implements com.borland.dx.dataset.Designable, com.borland.dx.dataset.LoadCancel, com.borland.dx.sql.dataset.ConnectionUpdateListener, com.borland.dx.sql.dataset.Task, java.io.Serializable, java.util.EventListener

The *OracleProcedureProvider* class provides data to the *StorageDataSet* by executing the specified stored procedure (Oracle PL-SQL) through JDBC. The *OracleProcedureProvider* class makes no attempt to make the *StorageDataSet* updatable or editable; it is the developer's responsibility to ensure this prior to the start of the resolution phase.

The procedure used for the *ProcedureDataSet* must have an OUT parameter of type CURSOR REF as the first parameter in the procedure specification (query string). Typically this will be a PL-SQL stored function with a return type of a CURSOR REF. Oracle JDBC drivers use an extension of jdbc for this data type.

OracleProcedureProvider properties

Property	Implemented in
accumulateResults	com.borland.dx.sql.dataset.JdbcProvider
class*	java.lang.Object
parameterRow	com.borland.dx.sql.dataset.ProcedureProvider
procedure	com.borland.dx.sql.dataset.ProcedureProvider

OracleProcedureProvider methods

Method	Implemented in
callProcedure (com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset. ReadWriteRow)	com.borland.dx.sql.dataset.ProcedureProvider
callProcedure(com.borland.dx.sql. dataset.Database, java.lang.String, com.borland.dx.dataset. ReadWriteRow[])	com.borland.dx.sql.dataset.ProcedureProvider
cancelLoad()	com.borland.dx.sql.dataset.JdbcProvider
canChangeConnection(com.borland. dx.sql.dataset. ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
checkIfBusy(com.borland.dx.dataset. StorageDataSet)	com.borland.dx.dataset.Provider
checkMasterLink (com.borland.dx.dataset. StorageDataSet, com.borland.dx.dataset. MasterLinkDescriptor)	com.borland.dx.dataset.Provider
clone()	java.lang.Object
close(com.borland.dx.dataset. StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
closeStatement()	com.borland.dx.sql.dataset.JdbcProvider
connectionChanged(com.borland.dx. sql.dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionClosed(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionOpening(com.borland.dx. sql.dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
equals(java.lang.Object)	java.lang.Object
executeTask()	com.borland.dx.sql.dataset.JdbcProvider
fetchDataSet()	com.borland.dx.sql.dataset.JdbcProvider
finalize()	java.lang.Object

Method	Implemented in
hashCode()	java.lang.Object
hasMoreData(com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
ifBusy()	com.borland.dx.sql.dataset.ProcedureProvider
ifBusy(com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
notify()	java.lang.Object
notifyAll()	java.lang.Object
provideData(com.borland.dx.dataset.StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
provideMoreData(com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ProcedureDataSet class

dx.sql.dataset package

Extends com.borland.dx.dataset.StorageDataSet

Implements com.borland.dx.dataset.AccessListener,
com.borland.dx.dataset.ColumnDesigner,
com.borland.dx.dataset.Designable,
com.borland.dx.dataset.MasterNavigateListener,
com.borland.dx.dataset.StatusListener, java.io.Serializable,
java.util.EventListener

The *ProcedureDataSet* class is an extension of its superclass (*StorageDataSet*) and provides functionality to run a stored procedure against data stored in a SQL database, passing in parameters if the procedure expects them. The procedure call is expected to return a cursor.

In any application that uses the *ProcedureDataSet*, the following components are also required:

- an instantiated *Database* component to handle the JDBC connection to the SQL database
- an instantiated *ProcedureDescriptor* object to store the stored procedure's properties

The data contained in a *ProcedureDataSet* is the result of the most recent execution of the stored procedure. Storing the “result set” from the execution of the stored procedure in a *ProcedureDataSet* allows for greater flexibility in navigation of the resulting data.

The *ProcedureDataSet* inherits the *maxRows* property which allows you to set the maximum number of rows stored initially in the *ProcedureDataSet* as a result of the stored procedure execution.

Once the data is stored in the *ProcedureDataSet*, you manipulate it and connect it to UI controls in exactly the same way as you would other *StorageDataSet* components, without regard to which component is storing the data.

This class is used with servers whose JDBC driver supports executing stored procedures that generate a result set. Not all JDBC drivers support this; some vendor libraries require special API calls to invoke stored procedures. Refer to your server documentation for more information on whether it meets this requirement.

Oracle stored procedures

Oracle stored procedures work only with Oracle's type-2 and type-4 drivers. Also, the Oracle server version must be 7.3.4 or 8.0.4 or newer.

Note Stored “Functions” work with older versions of Oracle Servers as well.

The following example demonstrates a stored procedure in a package:

```
CREATE PACKAGE my_pack is
    type cust_cursor is ref cursor return CUSTOMER%rowtype;
    procedure sp_test ( rc1 in out cust_cursor );
end;

CREATE PACKAGE BODY my_pack IS
    PROCEDURE sp_test (rc1 in out cust_cursor) IS
    BEGIN
        open rc1 for select * from CUSTOMER;
    END sp_test;
END my_pack;
```

The call string for this procedure should be:

```
"{ call my_pack.sp_test(?) }"
```

No parameter row is needed. JDataStore uses the cursor to load the *ProcedureDataSet*.

The result set you need to load the data into the *ProcedureDataSet* with must be the first parameter in the stored procedure argument list. If additional parameters need to be sent or received, specify the *ParameterRow* in the *Procedure* property (or programmatically through the *ProcedureDescriptor*).

Sybase stored procedures

Stored procedures on Sybase servers are created in a “Chained” transaction mode. In order to call Sybase stored procedures as part of a *ProcedureResolver*, the procedures must be modified to run in an unchained transaction mode. Use the Sybase stored system procedure *sp_procmode* to change the transaction mode to either “anymore” or “unchained”. See your Sybase documentation for additional information.

ProcedureDataSet properties

Property	Implemented in
accumulateResults	this class
allRowIds**	com.borland.dx.dataset.StorageDataSet
assignedNull**	com.borland.dx.dataset.ReadWriteRow
calcAggFieldsListener*	com.borland.dx.dataset.StorageDataSet
calcFieldsListener*	com.borland.dx.dataset.StorageDataSet
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns**	com.borland.dx.dataset.StorageDataSet
database*	this class
dataFile	com.borland.dx.dataset.StorageDataSet
defaultValues**	com.borland.dx.dataset.DataSet
deletedRowCount*	com.borland.dx.dataset.StorageDataSet
detailDataSetWithFetchAsNeeded*	com.borland.dx.dataset.DataSet
details*	com.borland.dx.dataset.DataSet
displayErrors	com.borland.dx.dataset.DataSet
duplicates*	com.borland.dx.dataset.StorageDataSet
editable	com.borland.dx.dataset.DataSet
editing*	com.borland.dx.dataset.DataSet
editingNewRow*	com.borland.dx.dataset.DataSet
empty*	com.borland.dx.dataset.DataSet
enableDelete	com.borland.dx.dataset.DataSet

Property	Implemented in
enableInsert	com.borland.dx.dataset.DataSet
enableUpdate	com.borland.dx.dataset.DataSet
insertedRowCount*	com.borland.dx.dataset.StorageDataSet
internalRow*	com.borland.dx.dataset.DataSet
lastColumnVisited	com.borland.dx.dataset.DataSet
locale	com.borland.dx.dataset.StorageDataSet
masterLink	com.borland.dx.dataset.DataSet
maxDesignRows	com.borland.dx.dataset.StorageDataSet
maxResolveErrors	com.borland.dx.dataset.StorageDataSet
maxRows	com.borland.dx.dataset.StorageDataSet
metaDataUpdate	com.borland.dx.dataset.StorageDataSet
needsRestructure*	com.borland.dx.dataset.StorageDataSet
open*	com.borland.dx.dataset.DataSet
parameterRow*	this class
procedure	this class
provider**	this class
queryString*	this class
readOnly	com.borland.dx.dataset.StorageDataSet
resolvable	com.borland.dx.dataset.StorageDataSet
resolveOrder	com.borland.dx.dataset.StorageDataSet
resolver	com.borland.dx.dataset.StorageDataSet
row*	com.borland.dx.dataset.DataSet
rowCount*	com.borland.dx.dataset.DataSet
rowFilterListener*	com.borland.dx.dataset.DataSet
schemaName	com.borland.dx.dataset.StorageDataSet
sort	com.borland.dx.dataset.DataSet
status*	com.borland.dx.dataset.DataSet
storageDataSet*	com.borland.dx.dataset.DataSet
store	com.borland.dx.dataset.StorageDataSet
storeClassFactory	com.borland.dx.dataset.StorageDataSet
storeName	com.borland.dx.dataset.StorageDataSet
tableName	com.borland.dx.dataset.StorageDataSet
unassignedNull**	com.borland.dx.dataset.ReadWriteRow
updatedRowCount*	com.borland.dx.dataset.StorageDataSet

accumulateResults

```
public final boolean isAccumulateResults()
public final void setAccumulateResults(boolean accumulate)
```

If **true**, the provided data is accumulated over consecutive calls to *executeQuery*. If **false**, subsequent *executeQuery* calls overwrite the existing *DataSet*.

database

```
public final Database getDatabase()
```

Read-only property that returns the *Database* object associated with this *ProcedureDataSet*. This property is a short cut to the *database* property of the *ProcedureDescriptor* object. Set this property using any *ProcedureDescriptor* constructor that takes a *Database* object as a parameter.

parameterRow

```
public ReadWriteRow getParameterRow()
```

Read-only property that returns the *ReadWriteRow* object associated with this *ProcedureDataSet*. This property is a short cut to the *parameterRow* property of the *ProcedureDescriptor* object. Set this property using any *ProcedureDescriptor* constructor that takes a *ReadWriteRow* object as a parameter.

procedure

```
public final ProcedureDescriptor getProcedure()
public final void setProcedure(ProcedureDescriptor procedureDescriptor)
```

The *ProcedureDescriptor* that stores the properties for this *ProcedureDataSet*.

provider

```
public void setProvider(Provider provider)
```

The provider for this *ProcedureDataSet*.

queryString

```
public final String getQueryString()
```

Read-only property that returns the query *String* associated with this *ProcedureDataSet*. This property is a short cut to the *queryString* property of the *ProcedureDescriptor* object. Set this property using any *ProcedureDescriptor* constructor that takes a query string as a parameter.

ProcedureDataSet methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	com.borland.dx.dataset.DataSet
addColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
addColumn(java.lang.String, int)	com.borland.dx.dataset.StorageDataSet
addColumn(java.lang.String, java.lang.String, int)	com.borland.dx.dataset.StorageDataSet
addLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
addRow(com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
addUniqueColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
allocateValues()	com.borland.dx.dataset.DataSet
atFirst()	com.borland.dx.dataset.DataSet
atLast()	com.borland.dx.dataset.DataSet
cancel()	com.borland.dx.dataset.DataSet
cancelLoading()	com.borland.dx.dataset.StorageDataSet
cancelOperation()	com.borland.dx.dataset.StorageDataSet
canNavigate (com.borland.dx.dataset.Column, int)	com.borland.dx.dataset.DataSet
canSet(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
changeColumn(int, com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
changesPending()	com.borland.dx.dataset.StorageDataSet
clearStatus()	com.borland.dx.dataset.DataSet
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneColumns()	com.borland.dx.dataset.StorageDataSet
cloneDataSetStructure()	com.borland.dx.dataset.StorageDataSet
cloneDataSetView()	com.borland.dx.dataset.DataSet
close()	com.borland.dx.dataset.DataSet
closeProvider(boolean)	com.borland.dx.dataset.StorageDataSet
closeStatement()	this class
columnIsVisible(java.lang.String)	com.borland.dx.dataset.DataSet
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	com.borland.dx.dataset.DataSet
deleteDuplicates()	com.borland.dx.dataset.StorageDataSet

Method	Implemented in
deleteRow()	com.borland.dx.dataset.DataSet
dittoRow(boolean, boolean)	com.borland.dx.dataset.DataSet
dittoRow(boolean)	com.borland.dx.dataset.DataSet
dropAllIndexes()	com.borland.dx.dataset.StorageDataSet
dropColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
dropColumn(java.lang.String)	com.borland.dx.dataset.StorageDataSet
dropIndex()	com.borland.dx.dataset.DataSet
dropIndex (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
editRow()	com.borland.dx.dataset.DataSet
empty()	com.borland.dx.dataset.StorageDataSet
emptyAllRows()	com.borland.dx.dataset.DataSet
emptyRow()	com.borland.dx.dataset.DataSet
enableDataSetEvents(boolean)	com.borland.dx.dataset.DataSet
endLoading()	com.borland.dx.dataset.StorageDataSet
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
executeQuery()	this class
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	com.borland.dx.dataset.DataSet
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow
getByte(java.lang.String)	com.borland.dx.dataset.ReadRow
getByteArray(int)	com.borland.dx.dataset.ReadRow
getByteArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getDataRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDataRow(int, com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDeletedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getDetail(java.lang.String)	com.borland.dx.dataset.DataSet
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInsertedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getOriginalRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.StorageDataSet
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getUpdatedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
goToClosestRow(int)	com.borland.dx.dataset.DataSet
goToInternalRow(long)	com.borland.dx.dataset.DataSet
goToRow (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
goToRow(int)	com.borland.dx.dataset.DataSet
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	com.borland.dx.dataset.DataSet
hashCode()	java.lang.Object
hasRowIds()	com.borland.dx.dataset.StorageDataSet
hasValidations()	com.borland.dx.dataset.DataSet
inBounds()	com.borland.dx.dataset.DataSet
indexExists (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
insertRow(boolean)	com.borland.dx.dataset.DataSet
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	com.borland.dx.dataset.DataSet
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.DataSet
isModified(java.lang.String)	com.borland.dx.dataset.DataSet
isNew(int)	com.borland.dx.dataset.DataSet
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	com.borland.dx.dataset.DataSet
loadRow()	com.borland.dx.dataset.StorageDataSet
loadRow(int)	com.borland.dx.dataset.StorageDataSet
locate(com.borland.dx.dataset.ReadRow, int)	com.borland.dx.dataset.DataSet
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	com.borland.dx.dataset.DataSet
masterNavigated (com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
masterNavigating (com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
moveColumn(int, int)	com.borland.dx.dataset.StorageDataSet

Method	Implemented in
moveRow(int)	com.borland.dx.dataset.DataSet
next()	com.borland.dx.dataset.DataSet
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.dx.dataset.DataSet
openDetails()	com.borland.dx.dataset.DataSet
post()	com.borland.dx.dataset.DataSet
postAllDataSets()	com.borland.dx.dataset.StorageDataSet
prior()	com.borland.dx.dataset.DataSet
provideMoreData()	com.borland.dx.dataset.StorageDataSet
recalc()	com.borland.dx.dataset.StorageDataSet
refetchRow (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.DataSet
refilter()	com.borland.dx.dataset.DataSet
refresh()	this class
refreshSupported()	this class
removeLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
reset()	com.borland.dx.dataset.StorageDataSet
resetInBounds()	com.borland.dx.dataset.DataSet
resetPendingStatus(boolean)	com.borland.dx.dataset.StorageDataSet
resetPendingStatus(long, boolean)	com.borland.dx.dataset.StorageDataSet
restructure()	com.borland.dx.dataset.StorageDataSet
saveChanges()	com.borland.dx.dataset.DataSet
saveChanges (com.borland.dx.dataset.DataSet)	this class
saveChangesSupported()	this class
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.DataSet

Method	Implemented in
setDisplayVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setRowId(java.lang.String, boolean)	com.borland.dx.dataset.StorageDataSet
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startEditCheck (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean)	com.borland.dx.dataset.StorageDataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean)	com.borland.dx.dataset.StorageDataSet

Method	Implemented in
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean)	com.borland.dx.dataset.StorageDataSet
statusMessage (com.borland.dx.dataset.StatusEvent)	com.borland.dx.dataset.DataSet
statusMessage(int, java.lang.String)	com.borland.dx.dataset.DataSet
toggleViewOrder(java.lang.String)	com.borland.dx.dataset.DataSet
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
validate()	com.borland.dx.dataset.DataSet
validate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

closeStatement()

public void closeStatement()

If *Database.isUseStatementCaching()* returns true, JDBC statements can be cached. By default these statements will be closed during garbage collection. If resources are scarce, the statement can be forced closed by calling this method.

executeQuery()

public final void executeQuery()

Executes the procedure with the specified *Database* and *procedure* properties, and populates the *DataSet*.

refresh()

public void refresh()

Executes the stored procedure and populates the *ProcedureDataSet*, using the settings for the *database* and *query* properties.

Overrides com.borland.dx.dataset.StorageDataSet.refresh()

refreshSupported()

public boolean refreshSupported()

Used internally to specify whether *refresh()* is always supported by the *ProcedureDataSet*.

Overrides com.borland.dx.dataset.StorageDataSet.refreshSupported()

saveChanges(com.borland.dx.dataset.DataSet)

```
public void saveChanges(DataSet dataSet)
```

Saves changes made to the data. If no resolver has been specified, a *QueryResolver* is used by default.

Overrides `com.borland.dx.dataset.StorageDataSet.saveChanges`
(`com.borland.dx.dataset.DataSet`)

saveChangesSupported()

```
public boolean saveChangesSupported()
```

Used internally to specify whether *saveChanges()* is always supported by the *ProcedureDataSet*.

Overrides `com.borland.dx.dataset.StorageDataSet.saveChangesSupported()`

ProcedureDataSet event listeners

This class is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

calcAggFields

```
public synchronized void addCalcAggFieldsListener(CalcAggFieldsListener listener)
public synchronized void removeCalcAggFieldsListener(CalcAggFieldsListener listener)
```

calcFields

```
public synchronized void addCalcFieldsListener(CalcFieldsListener listener)
public synchronized void removeCalcFieldsListener(CalcFieldsListener listener)
```

columnChange

```
public void addColumnChangeListener(ColumnChangeListener listener)
public synchronized void removeColumnChangeListener(ColumnChangeListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

edit

```
public void addEditListener(EditListener listener)
public synchronized void removeEditListener(EditListener listener)
```

load

```
public final synchronized void addLoadListener(LoadListener listener)
public final synchronized void removeLoadListener(LoadListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

ProcedureDescriptor class

dx.sql.dataset package

Extends com.borland.dx.sql.dataset.QueryDescriptor

Implements java.io.Serializable

The *ProcedureDescriptor* class stores property settings associated with a *ProcedureDataSet*. Its main properties are:

- its associated *Database* component (required)
- the stored procedure escape sequence or the SQL statement (required)

- whether to execute the stored procedure immediately when the *ProcedureDataSet* is opened (defaults to true)
- how the data should be loaded into the *ProcedureDataSet*

To work with this component programmatically, set its properties when instantiating the *ProcedureDescriptor* object, or individually by its write accessor methods. For properties that do not have corresponding setter methods, use a constructor that takes that property as a parameter.

Data can be loaded all in one fetch, as needed, asynchronously or one at a time. When working with asynchronous queries, opening the *ProcedureDataSet* then immediately calling methods such as *rowCount()* typically returns a row count lower than expected. To avoid this, either set the stored procedure to run synchronously, listen for the *LoadingEvent*, perform other actions while the *ProcedureDataSet* completes loading, or listen for updates to the row count. With asynchronous loading, as the stored procedure fetches rows of data, they are appended to the end of the *DataSet*. If working with a sorted view of the *DataSet*, the new rows appear in the specified sort order. Also, be careful to not make assumptions about the current row position since rows are inserted into the sorted view as they are fetched, thereby changing row positions automatically.

Note If a stored procedure is run against a synonym on an Oracle server, it is dependent on the support of synonyms in the JDBC driver to determine whether the stored procedure is updatable.

ProcedureDescriptor constructors

ProcedureDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String)

public ProcedureDescriptor(Database database, String query)

Constructs a *ProcedureDescriptor* object with the specified parameters:

<i>database</i>	The associated <i>Database</i> object.
<i>query</i>	The stored procedure escape or SQL statement to run against the database.

ProcedureDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean)

public ProcedureDescriptor(Database database, String query, ReadWriteRow parameters, boolean executeOnOpen)

Constructs a *ProcedureDescriptor* object with the specified parameters:

<i>database</i>	The associated <i>Database</i> object.
<i>query</i>	The stored procedure escape or SQL statement to run against the database.
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that stores the parameter values for the stored procedure.
<i>executeOnOpen</i>	Whether or not the stored procedure escape or SQL statement should execute immediately when an object that is bound to the <i>ProcedureDataSet</i> is opened.

ProcedureDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean, boolean)

public ProcedureDescriptor(Database database, String query, ReadWriteRow parameters, boolean executeOnOpen, boolean asynchronousExecution)

This constructor has been deprecated. Use a constructor that takes a *loadOption* parameter, or set the *loadOption* property directly.

ProcedureDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean, int)

public ProcedureDescriptor(Database database, String query, ReadWriteRow parameters, boolean executeOnOpen, int loadOption)

Constructs a *ProcedureDescriptor* object with the specified parameters:

<i>database</i>	The associated <i>Database</i> object.
<i>query</i>	The stored procedure escape or SQL statement to run against the database.
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that stores the parameter values for the stored procedure.
<i>executeOnOpen</i>	Whether or not the stored procedure escape or SQL statement should execute immediately when an object that is bound to the <i>ProcedureDataSet</i> is opened.
<i>loadOption</i>	How the data should be loaded into the <i>ProcedureDataSet</i> . Constants for this parameter are defined in <i>Load</i> variables.

ProcedureDescriptor properties

Property	Implemented in
asynchronousExecution	com.borland.dx.sql.dataset.QueryDescriptor
class*	java.lang.Object
database*	com.borland.dx.sql.dataset.QueryDescriptor
executeOnOpen	com.borland.dx.sql.dataset.QueryDescriptor
loadOption	com.borland.dx.sql.dataset.QueryDescriptor
parameterRow*	com.borland.dx.sql.dataset.QueryDescriptor
queryString*	com.borland.dx.sql.dataset.QueryDescriptor

ProcedureDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	com.borland.dx.sql.dataset.QueryDescriptor
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ProcedureProvider class

dx.sql.dataset package

Extends	com.borland.dx.sql.dataset.JdbcProvider
Extended by	com.borland.dx.sql.dataset.OracleProcedureProvider
Implements	com.borland.dx.dataset.Designable, com.borland.dx.dataset.LoadCancel, com.borland.dx.sql.dataset.ConnectionUpdateListener, com.borland.dx.sql.dataset.Task, java.io.Serializable, java.util.EventListener

The *ProcedureProvider* class provides data to the *StorageDataSet* by executing the specified stored procedure through JDBC. You connect this component to the *StorageDataSet* component through the *StorageDataSet*'s *provider* property.

This class provides the data to the *StorageDataSet* however it does not attempt to make the *StorageDataSet* updatable or editable; it is the

developer's responsibility to ensure this prior to the start of the resolution phase.

The stored procedure is expected to return a result set. For stored procedures that don't return values, use either static form of the *callProcedure()* method to execute them.

ProcedureProvider properties

Property	Implemented in
accumulateResults	com.borland.dx.sql.dataset.JdbcProvider
class*	java.lang.Object
parameterRow	this class
procedure	this class

parameterRow

```
public ReadWriteRow getParameterRow()
public void setParameterRow(ReadWriteRow value)
```

The *ReadWriteRow* that is used to fill in parameter values for parameterized queries or stored procedures for *StorageDataSet* extensions such as *QueryDataSet* and *ProcedureDataSet*.

procedure

```
public final ProcedureDescriptor getProcedure()
public final void setProcedure(ProcedureDescriptor procedureDescriptor)
```

Stores the *ProcedureDescriptor* object that contains property settings for the stored procedure.

ProcedureProvider methods

Method	Implemented in
callProcedure (com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)	this class
callProcedure (com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow[])	this class
cancelLoad()	com.borland.dx.sql.dataset.JdbcProvider
canChangeConnection(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider

Method	Implemented in
checkIfBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Provider
checkMasterLink (com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset. MasterLinkDescriptor)	com.borland.dx.dataset.Provider
clone()	java.lang.Object
close(com.borland.dx.dataset. StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
closeStatement()	com.borland.dx.sql.dataset.JdbcProvider
connectionChanged(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionClosed(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionOpening(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
equals(java.lang.Object)	java.lang.Object
executeTask()	com.borland.dx.sql.dataset.JdbcProvider
fetchDataSet()	com.borland.dx.sql.dataset.JdbcProvider
finalize()	java.lang.Object
hashCode()	java.lang.Object
hasMoreData (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
ifBusy()	this class
ifBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
notify()	java.lang.Object
notifyAll()	java.lang.Object
provideData (com.borland.dx.dataset.StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
provideMoreData (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

callProcedure(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)

public static final int callProcedure(Database database, String procedureSpecification, ReadWriteRow parameters)

Calls a stored procedure. For example,

```
callProcedure(db1, "call foo", null);
```

This method can also be used for queries with an output parameter, for example:

```
callProcedure(db1, "select sum(expense) into ? from expense_table", paramRow);
```

<i>database</i>	The associated <i>Database</i> object.
<i>procedureSpecification</i>	The procedure to call.
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that stores the parameter values for the stored procedure.

callProcedure(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow[])

public static final int callProcedure(Database database, String procedureSpecification, ReadWriteRow[] parameters)

Calls a stored procedure with named access to multiple rows of parameters. The parameters are accessed via an optional tag of the named parameters. The tag is the number of the passed *ReadWriteRow*, starting with 1. For example, the following code passes the value of the 'Name' column from the parameter row 'param1' as the first parameter, and the value of the 'Name' column from the parameter row 'param2'.

```
callProcedure(db1, "call foo(:2.Name, :1.Name)", new ReadWriteRow[]{param1, param2});
```

If no tag is given, the parameter name is found by searching the parameter rows from left to right. Therefore, tags can be used to differentiate columns with identical names in different parameter rows.

<i>database</i>	The associated <i>Database</i> object.
<i>procedureSpecification</i>	The procedure to call.
<i>parameters</i>	An array of <i>ReadWriteRow</i> objects containing the parameter values for the stored procedure.

ifBusy()

public void ifBusy()

Tests whether the data is present. This method is used when providing data asynchronously to determine whether editing, resolving, and other such actions should be blocked until the data is available.

ProcedureResolver component

dx.sql.dataset package

Extends com.borland.dx.sql.dataset.SQLResolver

Implements com.borland.dx.dataset.Designable, java.io.Serializable

The *ProcedureResolver* component is used to resolve (save) changes back to a JDBC data source by calling stored procedures in that database. The stored procedures must exist prior to using this component; this component will not generate them. These stored procedures must meet the requirements described in the following properties:

- *deleteProcedure*
- *insertProcedure*
- *updateProcedure*

The *database* property of this component must be set to the *Database* component that this *ProcedureResolver* is associated with. Otherwise, a *DataSetException* is generated.

Sybase users Stored procedures on Sybase servers are created in a “Chained” transaction mode. In order to call Sybase stored procedures as part of this component, the procedures must be modified to run in an unchained transaction mode. Use the Sybase stored system procedure *sp_procxmode* to change the transaction mode to either “anymore” or “unchained”. See your Sybase documentation for additional information.

ProcedureResolver constructors

ProcedureResolver()

public ProcedureResolver()

Constructs a *ProcedureResolver* component.

ProcedureResolver properties

Property	Implemented in
class*	java.lang.Object
database	this class
deleteProcedure	this class
insertProcedure	this class
updateProcedure	this class

database

```
public Database getDatabase()
public void setDatabase(Database database)
```

The *Database* object associated with this component. If not set, a *DataSetException* of *NO_DATABASE_TO_RESOLVE* is generated.

deleteProcedure

```
public ProcedureDescriptor getDeleteProcedure()
public void setDeleteProcedure(ProcedureDescriptor deleteProcedure)
```

Stores the *ProcedureDescriptor* (which includes the parameters used when calling the delete procedure) that is associated with the *ProcedureDataSet*. The *deleteProcedure* is invoked for every row, that was deleted in the *DataSet*.

The available parameters for invocation of a *deleteProcedure* call are:

- 1 the original row as it was when data was provided into the *DataSet*.
- 2 the optional *parameterRow* specified in the *ProcedureDescriptor*.

The stored procedure should be designed to delete a record in the appropriate table(s) given the original data of that row.

insertProcedure

```
public ProcedureDescriptor getInsertProcedure()
public void setInsertProcedure(ProcedureDescriptor insertProcedure)
```

Stores the *ProcedureDescriptor* (which includes the parameters used when calling the insert stored procedure) that is associated with the *ProcedureDataSet*. The *insertProcedure* is invoked for every row, that was inserted in the *DataSet*. The available parameters for invocation of an *insertProcedure* are:

- 1 the inserted row as it appears in the *DataSet*.
- 2 the optional *parameterRow* specified in the *ProcedureDescriptor*.

The stored procedure should be designed to insert a record in the appropriate table(s) given the data of that row. The *parameterRow* may be used for output summaries or optional input parameters.

updateProcedure

```
public ProcedureDescriptor getUpdateProcedure()
public void setUpdateProcedure(ProcedureDescriptor updateProcedure)
```

Stores the *ProcedureDescriptor* (which includes the parameters used when calling the update stored procedure) that is associated with the *ProcedureDataSet*. The *updateProcedure* is invoked for every row that was

changed in the *DataSet*. The available parameters for invocation of an *updateProcedure* are:

- 1 the modified row as it appears in the *DataSet*
- 2 the original row as it was when data was provided into the *DataSet*
- 3 the optional *parameterRow* specified in the *ProcedureDescriptor*

The stored procedure should be designed to update a record in the appropriate table(s) given the original data, and the modified data. Since the original row and the modified row have the same column names, the named parameter syntax has been expanded with a way to indicate the designated data row.

The named parameter “:2.CUST_ID” indicates the CUST_ID of the original data row, where “:1.CUST_ID” indicates the CUST_ID of the modified row.

ProcedureResolver methods

Method	Implemented in
checkIfBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Resolver
clone()	java.lang.Object
close (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Resolver
closeStatements (com.borland.dx.dataset.StorageDataSet)	this class
deleteRow(com.borland.dx.dataset.DataSet)	this class
equals(java.lang.Object)	java.lang.Object
fetchResolverListener()	com.borland.dx.sql.dataset.SQLResolver
finalize()	java.lang.Object
hashCode()	java.lang.Object
insertRow(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
resolveData (com.borland.dx.dataset.DataSet)	com.borland.dx.sql.dataset.SQLResolver
toString()	java.lang.Object
updateRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

closeStatements(com.borland.dx.dataset.StorageDataSet)

```
public void closeStatements(StorageDataSet dataSet)
```

Frees any system resources used for statements associated with the specified *StorageDataSet*.

Overrides `com.borland.dx.sql.dataset.SQLResolver.closeStatements`
(`com.borland.dx.dataset.StorageDataSet`)

deleteRow(com.borland.dx.dataset.DataSet)

```
public synchronized void deleteRow(DataSet dataSet)
```

Instructs the *Resolver* to delete the current row in the *DataSet* from the *Database*.

Overrides `com.borland.dx.sql.dataset.SQLResolver.deleteRow`
(`com.borland.dx.dataset.DataSet`)

insertRow(com.borland.dx.dataset.DataSet)

```
public synchronized void insertRow(DataSet dataSet)
```

Instructs the *Resolver* to insert the current row of the *DataSet* into the *Database*.

Overrides `com.borland.dx.sql.dataset.SQLResolver.insertRow`
(`com.borland.dx.dataset.DataSet`)

**updateRow(com.borland.dx.dataset.DataSet,
com.borland.dx.dataset.ReadWriteRow)**

```
public synchronized void updateRow(DataSet dataSet, ReadWriteRow oldDataRow)
```

Instructs the *Resolver* to update the current row of the *DataSet* in the *Database*.

Overrides `com.borland.dx.sql.dataset.SQLResolver.updateRow`
(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.ReadWriteRow`)

ProcedureResolver event listeners

This component is a source for the following event sets.

resolver

```
public synchronized void addResolverListener(ResolverListener listener)
```

```
public synchronized void removeResolverListener(ResolverListener listener)
```

QueryDataSet class

dx.sql.dataset package

Extends com.borland.dx.dataset.StorageDataSet

Implements com.borland.dx.dataset.AccessListener,
com.borland.dx.dataset.ColumnDesigner,
com.borland.dx.dataset.Designable,
com.borland.dx.dataset.MasterNavigateListener,
com.borland.dx.dataset.StatusListener, java.io.Serializable,
java.util.EventListener

The *QueryDataSet* class is an extension of its superclass (*StorageDataSet*) and provides functionality to run a query statement (with or without parameters) against a table in a SQL database.

In any application that uses the *QueryDataSet*, the following components are also required:

- an instantiated *Database* component to handle the JDBC connection to the SQL database
- an instantiated *QueryDescriptor* object to store the query properties

The data contained in a *QueryDataSet* is the result of the most recent query. The “result set” from the execution of the query is stored in the *QueryDataSet*, which allows for much greater flexibility in navigation of the resulting data. You specify how the data is loaded into the *QueryDataSet* (asynchronously, as needed, etc.) by specifying its *loadOption* property (stored in the associated *QueryDescriptor*). The *QueryDataSet* inherits the *maxRows* property which allows you to set the maximum number of rows that can be initially stored in the *QueryDataSet* from a query execution.

Once the data is stored in the *QueryDataSet*, you manipulate it and connect it to UI controls in exactly the same way as you would other *StorageDataSet* components, without regard to which component is storing the data.

The *QueryDataSet* component uses the *QueryProvider* and *QueryResolver* to perform the data providing and resolving functions.

Updatable versus read-only queries

By default, *JDataStore* automatically attempts to make a *QueryDataSet* updatable so that changes made to the data it contains can be resolved back to its data source. It analyzes the query and looks for row identifiers: one or more columns that uniquely identify each row. This is required to save any changed data back to the correct row of the original data. If you have not included these columns in your query’s SELECT statement, *JDataStore* automatically adds them but sets the visibility of such columns as “default”. This hides them from any data aware controls that this *QueryDataSet* is connected to. You may change the column’s visibility to **true** if desired.

Suppressing metadata discovery

To prevent the addition of row ID columns and various metadata related properties on *DataSet* and *Column* components, set this component's *metaDataUpdate* property to *MetaDataUpdate.NONE*.

If the *rowId* analysis fails or *metaDataUpdate* is set to *NONE*, you can make a *QueryDataSet* updatable by setting one or more of these properties (as applicable):

- Setting the *tableName* property of the *StorageDataSet* to the table name that is the data source of the *QueryDataSet*.
- Identifying a set of columns that can uniquely identify a row, for example, columns of a primary or secondary index. You set this using the *rowID* property of the *Column* components in the *StorageDataSet*. These row identifier columns are included in the query and the corresponding columns in the *QueryDataSet* are marked hidden by default.
- Setting the *readOnly* property to **false** (if not already).

Fine-tuning query performance

To improve *QueryDataSet* performance on data retrieval,

- For queries that return a small *ResultSet*, disabling the metadata discovery mechanisms for fetch operations can make a big performance improvement. Specifically, set the
 - *StorageDataSet.MetaDataUpdate* property to *MetaDataUpdate.NONE*.
 - *StorageDataSet.TableName* property to the table name.
 - *Column.RowId* property for the columns that uniquely and efficiently identify a row.

Note that the *QueryDataSet* only performs the metadata discovery operations the first time a query is run.

- Set the *LoadOption* property on the *QueryDataSet* or *ProcedureDataSet* to *Load.ASYNCHRONOUS* or *Load.AS_NEEDED*. You can also set this property to *Load.UNCACHED* if you will be reading the data one time in sequential order.
- For large result sets, using a *DataStore* can improve performance and save a lot of memory with its caching/persistence support.
- Statement caching. By default, DataExpress will cache prepared statements for both queries and stored procedures if *java.sql.Connection.getMetaData().getMaxStatements()* returns a value > 10. You can force statement caching by calling *Database.setCacheStatements(true)*. The prepared statements that are cached are not closed until one of the following happens:

- Some provider related property (for example, the *query* property) is changed.
- A *DataSet* component is garbage collected (the statement is closed in a *finalize()* method, *QueryDataSet.closeStatement()*, *ProcedureDataSet.closeStatement()*, *QueryProvider.closeStatement()* or *ProcedureProvider.closeStatement()*).

To improve performance when performing data inserts, deletes, and updates:

- For updates and deletes, set the *resolver* property to a *QueryResolver* and set the *updateMode* property of this *QueryResolver* to *UpdateMode.KEY_COLUMNS*. This weakens the optimistic concurrency used, but reduces the number of parameters set for an update/delete operation.
- For each call to *Database.saveChanges()*, calls are made to disable/enable a JDBC drivers autocommit mode. If your application calls *Database.saveChanges()* with the *useTransactions* parameter set to false, then these calls will not be made and the transaction will not be committed.
- By disabling the *resetPendingStatus* flag in the *Database.saveChanges()* method, further performance benefits can be achieved. With this disabled, DataExpress will not clear the *RowStatus* state for all inserted/deleted/updated rows. This is only desirable if you will not be calling *saveChanges()* with new edits on the *DataSet* without calling *refresh()* first.

Note that if transactions are disabled, your application must call *Database.commit()* or *Connection.commit()*.

Master-detail relationships

In a master-detail relationship, if you set the *fetchAsNeeded* property to **true**, you must include a WHERE clause in the detail query that matches the detail link column values to the master link column values. For more information on master-detail relationships, see *MasterLinkDescriptor*.

Alias support and column name conflicts

JDataStore supports column aliases that are specified in a SQL Select statement. The name specified as the alias is stored in the *columnName* property and is used to access the DataExpress API. The original data source name is stored in the *serverColumnName* property for use when resolving changes back to the data source.

When a query joins two or more tables, it may return several columns that have the same name (often the columns used to link the tables have the same name). JDataStore handles duplicate column names by appending a number ("EMP_NO", "EMP_No1", etc.). The modified column name is stored in the *columnName* property and is used to access the DataExpress API. The

original server name is stored in the *serverColumnName* property for (later) use when resolving data changes back to the data source.

These properties are also used for column aliases that are specified in SQL statements.

JDataStore also supports *table aliases*. For example, in the following query:

```
select e.emp_no, e.last_name empl_last, p.last_name, phone_last
  from employee e, phone_list p
 where e.emp_no = p.emp_no and
        e.last_name <> p.last_name
```

The employee table is assigned the alias “e” and its last_name column is given the alias “empl_last”. In multi-table queries, aliases can be useful.

Oracle synonyms

If a query is run against a synonym on an Oracle server, it is dependent on the support of synonyms in the JDBC driver to determine whether the query is updatable.

SQL views

Queries run against SQL views are supported, however, they may not be resolvable depending on what actions the SQL view performed. For example, if the view simply filters out rows, the server may be able to handle resolving the edits. You should be aware that you risk making edits that cannot later be resolved back to the data source. More likely, you will need to write your own resolver logic to handle this situation.

QueryDataSet properties

Property	Implemented in
accumulateResults	this class
allRowIds**	com.borland.dx.dataset.StorageDataSet
assignedNull**	com.borland.dx.dataset.ReadWriteRow
calcAggFieldsListener*	com.borland.dx.dataset.StorageDataSet
calcFieldsListener*	com.borland.dx.dataset.StorageDataSet
class*	java.lang.Object
columnCount*	com.borland.dx.dataset.ReadRow
columns**	com.borland.dx.dataset.StorageDataSet
database*	this class
dataFile	com.borland.dx.dataset.StorageDataSet
defaultValues**	com.borland.dx.dataset.DataSet
deletedRowCount*	com.borland.dx.dataset.StorageDataSet

Property	Implemented in
detailDataSetWithFetchAsNeeded*	com.borland.dx.dataset.DataSet
details*	com.borland.dx.dataset.DataSet
displayErrors	com.borland.dx.dataset.DataSet
duplicates*	com.borland.dx.dataset.StorageDataSet
editable	com.borland.dx.dataset.DataSet
editing*	com.borland.dx.dataset.DataSet
editingNewRow*	com.borland.dx.dataset.DataSet
empty*	com.borland.dx.dataset.DataSet
enableDelete	com.borland.dx.dataset.DataSet
enableInsert	com.borland.dx.dataset.DataSet
enableUpdate	com.borland.dx.dataset.DataSet
insertedRowCount*	com.borland.dx.dataset.StorageDataSet
internalRow*	com.borland.dx.dataset.DataSet
lastColumnVisited	com.borland.dx.dataset.DataSet
locale	com.borland.dx.dataset.StorageDataSet
masterLink	com.borland.dx.dataset.DataSet
maxDesignRows	com.borland.dx.dataset.StorageDataSet
maxResolveErrors	com.borland.dx.dataset.StorageDataSet
maxRows	com.borland.dx.dataset.StorageDataSet
metaDataUpdate	com.borland.dx.dataset.StorageDataSet
needsRestructure*	com.borland.dx.dataset.StorageDataSet
open*	com.borland.dx.dataset.DataSet
originalQueryString*	this class
parameterRow*	this class
provider**	this class
query	this class
readOnly	com.borland.dx.dataset.StorageDataSet
resolvable	com.borland.dx.dataset.StorageDataSet
resolveOrder	com.borland.dx.dataset.StorageDataSet
resolver	com.borland.dx.dataset.StorageDataSet
row*	com.borland.dx.dataset.DataSet
rowCount*	com.borland.dx.dataset.DataSet
rowFilterListener*	com.borland.dx.dataset.DataSet
schemaName	com.borland.dx.dataset.StorageDataSet
sort	com.borland.dx.dataset.DataSet
status*	com.borland.dx.dataset.DataSet
storageDataSet*	com.borland.dx.dataset.DataSet
store	com.borland.dx.dataset.StorageDataSet
storeClassFactory	com.borland.dx.dataset.StorageDataSet
storeName	com.borland.dx.dataset.StorageDataSet
tableName	com.borland.dx.dataset.StorageDataSet

Property	Implemented in
unassignedNull**	com.borland.dx.dataset.ReadWriteRow
updatedRowCount*	com.borland.dx.dataset.StorageDataSet

accumulateResults

```
public final boolean isAccumulateResults()
public final void setAccumulateResults(boolean accumulate)
```

Determines whether the provided data is accumulated over consecutive calls to the *executeQuery()* method (true) or not (false). If this property is disabled, subsequent *executeQuery()* calls overwrite the existing *DataSet*.

database

```
public final Database getDatabase()
```

Read-only property that stores the *Database* object that holds the connection to the SQL server.

originalQueryString

```
public final String getOriginalQueryString()
```

Read-only property that returns the original query string. The query string may be automatically updated by *JDataStore* to include columns that form a unique rowID. Use this read accessor to obtain the original, unaltered query string.

parameterRow

```
public ReadWriteRow getParameterRow()
```

Read-only property that returns the *ReadWriteRow* object that contains the parameters for the query.

Note If the query contains parameters, you need to call *DataSet.close()* before providing for another *QueryDataSet*.

provider

```
public void setProvider(Provider provider)
```

Stores the *QueryProvider* associated with this *QueryDataSet* component. When setting this property, only a *QueryProvider* is acceptable. A *DataSetException* of *NEED_QUERY_PROVIDER* is generated if this property is not set.

query

```
public final QueryDescriptor getQuery()  
public final void setQuery(QueryDescriptor queryDescriptor)
```

Specifies the QueryDescriptor object that stores query properties which make the QueryDataSet usable.

QueryDataSet methods

Method	Implemented in
accessChange (com.borland.dx.dataset.AccessEvent)	com.borland.dx.dataset.DataSet
addColumn(com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
addColumn(java.lang.String, int)	com.borland.dx.dataset.StorageDataSet
addColumn(java.lang.String, java.lang.String, int)	com.borland.dx.dataset.StorageDataSet
addLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
addRow(com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
addUniqueColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
allocateValues()	com.borland.dx.dataset.DataSet
atFirst()	com.borland.dx.dataset.DataSet
atLast()	com.borland.dx.dataset.DataSet
cancel()	com.borland.dx.dataset.DataSet
cancelLoading()	com.borland.dx.dataset.StorageDataSet
cancelOperation()	com.borland.dx.dataset.StorageDataSet
canNavigate (com.borland.dx.dataset.Column, int)	com.borland.dx.dataset.DataSet
canSet(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
changeColumn(int, com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
changesPending()	com.borland.dx.dataset.StorageDataSet
clearStatus()	com.borland.dx.dataset.DataSet
clearValues()	com.borland.dx.dataset.ReadWriteRow
clone()	java.lang.Object
cloneColumns()	com.borland.dx.dataset.StorageDataSet
cloneDataSetStructure()	com.borland.dx.dataset.StorageDataSet
cloneDataSetView()	com.borland.dx.dataset.DataSet
close()	com.borland.dx.dataset.DataSet
closeProvider(boolean)	com.borland.dx.dataset.StorageDataSet
closeStatement()	this class
columnIsVisible(java.lang.String)	com.borland.dx.dataset.DataSet

Method	Implemented in
copyTo (com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
copyTo(java.lang.String[], com.borland.dx.dataset.ReadRow, java.lang.String[], com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.ReadRow
deleteAllRows()	com.borland.dx.dataset.DataSet
deleteDuplicates()	com.borland.dx.dataset.StorageDataSet
deleteRow()	com.borland.dx.dataset.DataSet
dittoRow(boolean, boolean)	com.borland.dx.dataset.DataSet
dittoRow(boolean)	com.borland.dx.dataset.DataSet
dropAllIndexes()	com.borland.dx.dataset.StorageDataSet
dropColumn (com.borland.dx.dataset.Column)	com.borland.dx.dataset.StorageDataSet
dropColumn(java.lang.String)	com.borland.dx.dataset.StorageDataSet
dropIndex()	com.borland.dx.dataset.DataSet
dropIndex (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
editRow()	com.borland.dx.dataset.DataSet
empty()	com.borland.dx.dataset.StorageDataSet
emptyAllRows()	com.borland.dx.dataset.DataSet
emptyRow()	com.borland.dx.dataset.DataSet
enableDataSetEvents(boolean)	com.borland.dx.dataset.DataSet
endLoading()	com.borland.dx.dataset.StorageDataSet
equals(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
equals(java.lang.Object)	java.lang.Object
executeQuery()	this class
finalize()	java.lang.Object
findDifference(int, com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
findModified(int)	com.borland.dx.dataset.ReadRow
findOrdinal(java.lang.String)	com.borland.dx.dataset.ReadRow
first()	com.borland.dx.dataset.DataSet
format(int)	com.borland.dx.dataset.ReadRow
format(java.lang.String)	com.borland.dx.dataset.ReadRow
getArrayLength(java.lang.String)	com.borland.dx.dataset.ReadRow
getBigDecimal(int)	com.borland.dx.dataset.ReadRow
getBigDecimal(java.lang.String)	com.borland.dx.dataset.ReadRow
getBinaryStream(int)	com.borland.dx.dataset.ReadRow
getBoolean(int)	com.borland.dx.dataset.ReadRow
getBoolean(java.lang.String)	com.borland.dx.dataset.ReadRow
getByte(int)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getBytes(java.lang.String)	com.borland.dx.dataset.ReadRow
getBytesArray(int)	com.borland.dx.dataset.ReadRow
getBytesArray(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumn(int)	com.borland.dx.dataset.ReadRow
getColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
getColumnNames(int)	com.borland.dx.dataset.ReadRow
getDataRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDataRow(int, com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
getDate(int)	com.borland.dx.dataset.ReadRow
getDate(java.lang.String)	com.borland.dx.dataset.ReadRow
getDeletedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getDetail(java.lang.String)	com.borland.dx.dataset.DataSet
getDisplayVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getDouble(int)	com.borland.dx.dataset.ReadRow
getDouble(java.lang.String)	com.borland.dx.dataset.ReadRow
getFloat(int)	com.borland.dx.dataset.ReadRow
getFloat(java.lang.String)	com.borland.dx.dataset.ReadRow
getInputStream(int)	com.borland.dx.dataset.ReadRow
getInputStream(java.lang.String)	com.borland.dx.dataset.ReadRow
getInsertedRows(com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getInt(int)	com.borland.dx.dataset.ReadRow
getInt(java.lang.String)	com.borland.dx.dataset.ReadRow
getLong(int)	com.borland.dx.dataset.ReadRow
getLong(java.lang.String)	com.borland.dx.dataset.ReadRow
getObject(int)	com.borland.dx.dataset.ReadRow
getObject(java.lang.String)	com.borland.dx.dataset.ReadRow
getOriginalRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	com.borland.dx.dataset.StorageDataSet
getQueryString (com.borland.dx.dataset.StorageDataSet)	this class
getShort(int)	com.borland.dx.dataset.ReadRow
getShort(java.lang.String)	com.borland.dx.dataset.ReadRow
getString(int)	com.borland.dx.dataset.ReadRow
getString(java.lang.String)	com.borland.dx.dataset.ReadRow
getTime(int)	com.borland.dx.dataset.ReadRow
getTime(java.lang.String)	com.borland.dx.dataset.ReadRow
getTimestamp(int)	com.borland.dx.dataset.ReadRow

Method	Implemented in
getTimestamp(java.lang.String)	com.borland.dx.dataset.ReadRow
getUpdatedRows (com.borland.dx.dataset.DataSetView)	com.borland.dx.dataset.StorageDataSet
getVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(int, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
getVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadRow
getVariant(java.lang.String, int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
goToClosestRow(int)	com.borland.dx.dataset.DataSet
goToInternalRow(long)	com.borland.dx.dataset.DataSet
goToRow(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
goToRow(int)	com.borland.dx.dataset.DataSet
hasColumn(java.lang.String)	com.borland.dx.dataset.ReadRow
hasDetail(java.lang.String)	com.borland.dx.dataset.DataSet
hashCode()	java.lang.Object
hasRowIds()	com.borland.dx.dataset.StorageDataSet
hasValidations()	com.borland.dx.dataset.DataSet
inBounds()	com.borland.dx.dataset.DataSet
indexExists (com.borland.dx.dataset.SortDescriptor, com.borland.dx.dataset.RowFilterListener)	com.borland.dx.dataset.StorageDataSet
insertRow(boolean)	com.borland.dx.dataset.DataSet
interactiveLocate(java.lang.String, java.lang.String, int, boolean)	com.borland.dx.dataset.DataSet
isAssignedNull(int)	com.borland.dx.dataset.ReadRow
isAssignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isCompatibleList (com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.ReadRow
isModified(int)	com.borland.dx.dataset.DataSet
isModified(java.lang.String)	com.borland.dx.dataset.DataSet
isNew(int)	com.borland.dx.dataset.DataSet
isNull(int)	com.borland.dx.dataset.ReadRow
isNull(java.lang.String)	com.borland.dx.dataset.ReadRow
isUnassignedNull(int)	com.borland.dx.dataset.ReadRow
isUnassignedNull(java.lang.String)	com.borland.dx.dataset.ReadRow
last()	com.borland.dx.dataset.DataSet
loadRow()	com.borland.dx.dataset.StorageDataSet
loadRow(int)	com.borland.dx.dataset.StorageDataSet
locate(com.borland.dx.dataset.ReadRow, int)	com.borland.dx.dataset.DataSet

Method	Implemented in
lookup(com.borland.dx.dataset.ReadRow, com.borland.dx.dataset.DataRow, int)	com.borland.dx.dataset.DataSet
masterNavigated (com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
masterNavigating (com.borland.dx.dataset. MasterNavigateEvent)	com.borland.dx.dataset.DataSet
moveColumn(int, int)	com.borland.dx.dataset.StorageDataSet
moveRow(int)	com.borland.dx.dataset.DataSet
next()	com.borland.dx.dataset.DataSet
notify()	java.lang.Object
notifyAll()	java.lang.Object
open()	com.borland.dx.dataset.DataSet
openDetails()	com.borland.dx.dataset.DataSet
post()	com.borland.dx.dataset.DataSet
postAllDataSets()	com.borland.dx.dataset.StorageDataSet
prior()	com.borland.dx.dataset.DataSet
provideMoreData()	com.borland.dx.dataset.StorageDataSet
recalc()	com.borland.dx.dataset.StorageDataSet
refetchRow (com.borland.dx.dataset.ReadWriteRow)	this class
refilter()	com.borland.dx.dataset.DataSet
refresh()	this class
refreshSupported()	this class
removeLoadRowListener(listener)	com.borland.dx.dataset.StorageDataSet
requiredColumnsCheck()	com.borland.dx.dataset.ReadWriteRow
reset()	com.borland.dx.dataset.StorageDataSet
resetInBounds()	com.borland.dx.dataset.DataSet
resetPendingStatus(boolean)	com.borland.dx.dataset.StorageDataSet
resetPendingStatus(long, boolean)	com.borland.dx.dataset.StorageDataSet
restructure()	com.borland.dx.dataset.StorageDataSet
saveChanges()	com.borland.dx.dataset.DataSet
saveChanges (com.borland.dx.dataset.DataSet)	this class
saveChangesSupported()	this class
setBigDecimal(int, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBigDecimal(java.lang.String, java.math.BigDecimal)	com.borland.dx.dataset.ReadWriteRow
setBoolean(int, boolean)	com.borland.dx.dataset.ReadWriteRow
setBoolean(java.lang.String, boolean)	com.borland.dx.dataset.ReadWriteRow
setByte(int, byte)	com.borland.dx.dataset.ReadWriteRow
setByte(java.lang.String, byte)	com.borland.dx.dataset.ReadWriteRow

Method	Implemented in
setByteArray(int, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setByteArray(java.lang.String, byte[], int)	com.borland.dx.dataset.ReadWriteRow
setDate(int, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(int, long)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, java.sql.Date)	com.borland.dx.dataset.ReadWriteRow
setDate(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setDefaultValues()	com.borland.dx.dataset.DataSet
setDisplayVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.DataSet
setDouble(int, double)	com.borland.dx.dataset.ReadWriteRow
setDouble(java.lang.String, double)	com.borland.dx.dataset.ReadWriteRow
setFloat(int, float)	com.borland.dx.dataset.ReadWriteRow
setFloat(java.lang.String, float)	com.borland.dx.dataset.ReadWriteRow
setInputStream(int, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInputStream(java.lang.String, java.io.InputStream)	com.borland.dx.dataset.ReadWriteRow
setInt(int, int)	com.borland.dx.dataset.ReadWriteRow
setInt(java.lang.String, int)	com.borland.dx.dataset.ReadWriteRow
setLong(int, long)	com.borland.dx.dataset.ReadWriteRow
setLong(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setObject(int, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setObject(java.lang.String, java.lang.Object)	com.borland.dx.dataset.ReadWriteRow
setRowId(java.lang.String, boolean)	com.borland.dx.dataset.StorageDataSet
setShort(int, short)	com.borland.dx.dataset.ReadWriteRow
setShort(java.lang.String, short)	com.borland.dx.dataset.ReadWriteRow
setString(int, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setString(java.lang.String, java.lang.String)	com.borland.dx.dataset.ReadWriteRow
setTime(int, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(int, long)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, java.sql.Time)	com.borland.dx.dataset.ReadWriteRow
setTime(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(int, long)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, java.sql.Timestamp)	com.borland.dx.dataset.ReadWriteRow
setTimestamp(java.lang.String, long)	com.borland.dx.dataset.ReadWriteRow
setVariant(int, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
setVariant(java.lang.String, com.borland.dx.dataset.Variant)	com.borland.dx.dataset.ReadWriteRow
startEdit(com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet

Method	Implemented in
startEditCheck (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean, boolean)	com.borland.dx.dataset.StorageDataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean, boolean)	com.borland.dx.dataset.StorageDataSet
startLoading (com.borland.dx.dataset.LoadCancel, int, boolean)	com.borland.dx.dataset.StorageDataSet
statusMessage (com.borland.dx.dataset.StatusEvent)	com.borland.dx.dataset.DataSet
statusMessage(int, java.lang.String)	com.borland.dx.dataset.DataSet
toggleViewOrder(java.lang.String)	com.borland.dx.dataset.DataSet
toString()	com.borland.dx.dataset.ReadRow
updateRow (com.borland.dx.dataset.DataRow)	com.borland.dx.dataset.DataSet
validate()	com.borland.dx.dataset.DataSet
validate(com.borland.dx.dataset.ReadRow)	com.borland.dx.dataset.DataSet
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

closeStatement()

public void closeStatement()

If *Database.isUseStatementCaching()* returns true, JDBC statements can be cached. By default these statements will be closed during garbage collection. If resources are scarce, the statement can be forced closed by calling this method.

executeQuery()

public final void executeQuery()

Calls refresh().

You can call the *executeQuery()* method without an open *DataSet*. However, if the *DataSet* is open, *executeQuery()* will close the *DataSet*, execute the query, then re-open the *DataSet*.

getQueryString(com.borland.dx.dataset.StorageDataSet)

```
public final String getQueryString(StorageDataSet sds)
```

Returns the query string executed against the data source. This query string may have been automatically altered by JDataStore; use *getOriginalQueryString()* to obtain the original, unaltered query string.

refetchRow(com.borland.dx.dataset.ReadWriteRow)

```
public void refetchRow(ReadWriteRow row)
```

Fetches the original row from the data source based on the key field of the row you pass in. For example, if the key field of the row is “foobar”, this method fetches the row in the *DataSet* with that key field.

Overrides com.borland.dx.dataset.DataSet.refetchRow(ReadWriteRow)

refresh()

```
public void refresh()
```

Given that the database and query properties have been set, executes the query and populates the *DataSet*.

Overrides com.borland.dx.dataset.StorageDataSet.refresh()

refreshSupported()

```
public boolean refreshSupported()
```

Returns **true**.

Overrides com.borland.dx.dataset.StorageDataSet.refreshSupported()

saveChanges(com.borland.dx.dataset.DataSet)

```
public void saveChanges(DataSet dataSet)
```

Calls *Database.saveChanges()* with the *DataSet* object specified in its parameter.

Overrides com.borland.dx.dataset.StorageDataSet.saveChanges
(com.borland.dx.dataset.DataSet)

saveChangesSupported()

```
public boolean saveChangesSupported()
```

Returns true. Used internally by data-aware controls to determine if a *saveChanges()* type operation is supported.

Overrides com.borland.dx.dataset.StorageDataSet.saveChangesSupported()

QueryDataSet event listeners

This class is a source for the following event sets.

access

```
public final void addAccessListener(AccessListener listener)
public final void removeAccessListener(AccessListener listener)
```

calcAggFields

```
public synchronized void addCalcAggFieldsListener(CalcAggFieldsListener listener)
public synchronized void removeCalcAggFieldsListener(CalcAggFieldsListener listener)
```

calcFields

```
public synchronized void addCalcFieldsListener(CalcFieldsListener listener)
public synchronized void removeCalcFieldsListener(CalcFieldsListener listener)
```

columnChange

```
public void addColumnChangeListener(ColumnChangeListener listener)
public synchronized void removeColumnChangeListener(ColumnChangeListener listener)
```

dataChange

```
public final void addDataChangeListener(DataChangeListener listener)
public final void removeDataChangeListener(DataChangeListener listener)
```

edit

```
public void addEditListener(EditListener listener)
public synchronized void removeEditListener(EditListener listener)
```

load

```
public final synchronized void addLoadListener(LoadListener listener)
public final synchronized void removeLoadListener(LoadListener listener)
```

masterNavigate

```
public final void addMasterNavigateListener(MasterNavigateListener listener)
public final void removeMasterNavigateListener(MasterNavigateListener listener)
```

navigation

```
public final void addNavigationListener(NavigationListener listener)
public final void removeNavigationListener(NavigationListener listener)
```

open

```
public final void addOpenListener(OpenListener listener)
public final void removeOpenListener(OpenListener listener)
```

rowFilter

```
public final void addRowFilterListener(RowFilterListener listener)
public final void removeRowFilterListener(RowFilterListener listener)
```

status

```
public final void addStatusListener(StatusListener listener)
public final void removeStatusListener(StatusListener listener)
```

QueryDescriptor class

dx.sql.dataset package

Extends java.lang.Object

Extended by com.borland.dx.sql.dataset.ProcedureDescriptor

Implements java.io.Serializable

The *QueryDescriptor* stores properties that set a query statement to run against a SQL database. To access SQL table data using a *QueryDataSet*, the *Database* component and *QueryDescriptor* object are also required. The properties of the *QueryDescriptor* object are:

- the *Database* object which is associated to a particular *userName* and *connectionURL* (required)
- the query string to execute (required)
- whether the query executes automatically when the *QueryDataSet* is opened.
- whether to resource the query string to a separate *ResourceBundle*
- how the data is loaded into the *QueryDataSet*
- the list of query parameters

To work with this component programmatically, you set its properties when instantiating the *QueryDescriptor* object, or individually by its write accessor methods. For properties that do not have a corresponding setter method, use a constructor that takes that property as a parameter.

Loading data

Data can be loaded all in one fetch, as needed, asynchronously or one at a time (as specified by the *loadOption* property). When working with asynchronous queries, opening the *QueryDataSet* then immediately calling methods such as *rowCount()* typically returns a row count lower than expected. To avoid this, either set the query to synchronous, listen for the *LoadingEvent*, perform other actions while the *QueryDataSet* completes loading, or listen for updates to the row count.

As an asynchronous query fetches rows of data, they are appended to the end of the *DataSet*. When working with a sorted view of the *DataSet*, the new rows appear in the specified sort order. Also, be careful to not make assumptions about the current row position since rows are inserted into the sorted view as they are fetched, thereby changing row positions automatically.

Executing queries

When executing queries, you have several options from which to choose, depending on your query statement. If your statement returns a *ResultSet*, use a *QueryDataSet* component with a *QueryDescriptor*. For example, the statement

```
select * from customer
```

If you also require the addition of parameter passing (all named or all unnamed), assign these values to an object derived from *ReadWriteRow* (for example, a *ParameterRow*) in conjunction with the *QueryDataSet*.

Do not use a *QueryDataSet* if your SQL statement doesn't yield a *ResultSet*. Instead, use one of the following:

- If your statement doesn't require parameter passing use the *Database.executeStatement(...)* method.
- To execute a SQL statement with parameters, use the *QueryProvider.executeStatement(...)* method.
- To execute a statement that has output parameters, use the *ProcedureProvider.callProcedure(...)* method.

Resourceable SQL

Through the Query property editor, the SQL statement entered may be optionally resourced into a separate file. This provides a logical separation between the code which uses the SQL statement and the contents of that statement. This allows a developer to change the SQL statement inside the resourced file without needing to recompile the code which uses the SQL.

Oracle synonyms

If a query is run against a synonym on an Oracle server, it is dependent on the support of synonyms in the JDBC driver to determine whether the query is updatable.

QueryDescriptor constructors

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String)

public QueryDescriptor(Database database, String query)

Constructs a *QueryDescriptor* object with the following properties:

<i>database</i>	The <i>Database</i> object the <i>QueryDataSet</i> is to run against.
<i>query</i>	The query string to execute against the specified <i>database</i> .

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, boolean)

public QueryDescriptor(Database database, String query, boolean executeOnOpen)

Constructs a *QueryDescriptor* object with the following properties:

<i>database</i>	The <i>Database</i> object the <i>QueryDataSet</i> is to run against.
<i>query</i>	The query string to execute against the specified <i>database</i> .
<i>executeOnOpen</i>	Whether or not the query should execute immediately when an object that is bound to the <i>QueryDataSet</i> is opened.

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)

public QueryDescriptor(Database database, String query, ReadWriteRow parameters)

Constructs a *QueryDescriptor* object with the following properties:

<i>database</i>	The <i>Database</i> object the <i>QueryDataSet</i> is to run against.
<i>query</i>	The query string to execute against the specified <i>database</i> .
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that contains the values to use for the parameters of the query string.

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean)

```
public QueryDescriptor(Database database, String query, ReadWriteRow parameters, boolean
    executeOnOpen)
```

Constructs a *QueryDescriptor* object with the following properties:

<i>database</i>	The <i>Database</i> object the <i>QueryDataSet</i> is to run against.
<i>query</i>	The query string to execute against the specified <i>database</i> .
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that contains the values to use for the parameters of the query string.
<i>executeOnOpen</i>	Whether or not the query should execute immediately when an object that is bound to the <i>QueryDataSet</i> is opened.

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean, boolean)

```
public QueryDescriptor(Database database, String query, ReadWriteRow parameters, boolean
    executeOnOpen, boolean asynchronousExecution)
```

This constructor has been deprecated. Use a constructor that takes a *loadOption* parameter, or set the *loadOption* property directly.

QueryDescriptor(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow, boolean, int)

```
public QueryDescriptor(Database database, String query, ReadWriteRow parameters, boolean
    executeOnOpen, int loadOption)
```

Constructs a *QueryDescriptor* object with the following properties:

<i>database</i>	The <i>Database</i> object the <i>QueryDataSet</i> is to run against.
<i>query</i>	The query string to execute against the specified <i>database</i> .
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that contains the values to use for the parameters of the query string.

<i>executeOnOpen</i>	Whether or not the query should execute immediately when an object that is bound to the <i>QueryDataSet</i> is opened.
<i>loadOption</i>	How the data should be loaded into the <i>QueryDataSet</i> . Constants for this parameter are defined as <i>Load</i> variables.

QueryDescriptor properties

Property	Implemented in
<i>asynchronousExecution</i>	this class
<i>class*</i>	java.lang.Object
<i>database*</i>	this class
<i>executeOnOpen</i>	this class
<i>loadOption</i>	this class
<i>parameterRow*</i>	this class
<i>queryString*</i>	this class

asynchronousExecution

```
public final boolean isAsynchronousExecution()
public final void setAsynchronousExecution(boolean async)
```

This property has been deprecated. Use a constructor that takes a *loadOption* parameter, or set the *loadOption* property directly.

database

```
public Database getDatabase()
```

Read-only property that returns the instantiated Database that the query is associated with.

To set this property, use any *QueryDescriptor* constructor that takes this property as a parameter.

executeOnOpen

```
public boolean isExecuteOnOpen()
public void setExecuteOnOpen(boolean executeOnOpen)
```

If the *QueryDataSet* is empty, this property specifies whether the query is executed automatically when the *QueryDataSet* is opened. (“Empty” refers to a *DataSet* not having any visible or non-visible rows.) When **true** (the default value), the query is executed automatically and allows for live data to display in UI controls (such as the *GridControl*) in the JBuilder UI Designer.

To prevent the query from automatically executing when the *QueryDataSet* is opened, set this property to **false**.

This property does not apply to the *ProcedureDescriptor* subclass of the *QueryDescriptor* when used with a *ProcedureResolver* component.

loadOption

```
public final int getLoadOption()
public final void setLoadOption(int loadOption)
```

Returns the options specified for loading the *QueryDataSet* with data. Valid values for this property are defined in *Load* variables.

parameterRow

```
public ReadWriteRow getParameterRow()
```

Specifies the *ReadWriteRow* implementation that stores the parameters for the query statement.

queryString

```
public String getQueryString()
```

Read-only property that returns the query string to run against the *Database*.
To set this property, use any *QueryDescriptor* constructor that takes this property as a parameter.

QueryDescriptor methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

toString()

```
public String toString()
```

Returns a *String* equivalent of the property values stored in the *QueryDescriptor*.

Overrides `java.lang.Object.toString()`

QueryProvider component

dx.sql.dataset package

Extends `com.borland.dx.sql.dataset.JdbcProvider`

Implements `com.borland.dx.dataset.Designable`, `com.borland.dx.dataset.LoadCancel`, `com.borland.dx.sql.dataset.ConnectionUpdateListener`, `com.borland.dx.sql.dataset.Task`, `java.io.Serializable`, `java.util.EventListener`

The *QueryProvider* component is used to provide data to a *DataSet* by running a query through JDBC. This component is also a place holder for static methods for executing statements with parameters (see *executeStatement()*).

For information on performance tuning when executing queries, see “Fine tuning query performance” in the About section of the *QueryDataSet* component.

QueryProvider constructors

QueryProvider()

```
public QueryProvider()
```

Creates a *QueryProvider* object.

QueryProvider properties

Property	Implemented in
accumulateResults	<code>com.borland.dx.sql.dataset.JdbcProvider</code>
class*	<code>java.lang.Object</code>
parameterRow	this class
query	this class

parameterRow

```
public ReadWriteRow getParameterRow()
public void setParameterRow(ReadWriteRow value)
```

The *ReadWriteRow* that is used to fill in parameter values for parameterized queries or stored procedures for *StorageDataSet* extensions such as *QueryDataSet* and *ProcedureDataSet*.

query

```
public final QueryDescriptor getQuery()
public final void setQuery(QueryDescriptor queryDescriptor)
```

The *QueryDescriptor* object that contains query properties.

QueryProvider methods

Method	Implemented in
cancelLoad()	com.borland.dx.sql.dataset.JdbcProvider
canChangeConnection (com.borland.dx.sql.dataset. ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
checkIfBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Provider
checkMasterLink (com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset. MasterLinkDescriptor)	this class
clone()	java.lang.Object
close (com.borland.dx.dataset.StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
closeStatement()	com.borland.dx.sql.dataset.JdbcProvider
connectionChanged (com.borland.dx.sql.dataset. ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionClosed(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
connectionOpening(com.borland.dx.sql. dataset.ConnectionUpdateEvent)	com.borland.dx.sql.dataset.JdbcProvider
equals(java.lang.Object)	java.lang.Object
executeStatement (com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)	this class

Method	Implemented in
executeStatement (com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow[])	this class
executeTask()	com.borland.dx.sql.dataset.JdbcProvider
fetchDataSet()	com.borland.dx.sql.dataset.JdbcProvider
finalize()	java.lang.Object
getQueryString (com.borland.dx.dataset.StorageDataSet)	this class
hashCode()	java.lang.Object
hasMoreData (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
ifBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
notify()	java.lang.Object
notifyAll()	java.lang.Object
provideData (com.borland.dx.dataset.StorageDataSet, boolean)	com.borland.dx.sql.dataset.JdbcProvider
provideMoreData (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.sql.dataset.JdbcProvider
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

checkMasterLink(com.borland.dx.dataset.StorageDataSet, com.borland.dx.dataset.MasterLinkDescriptor)

public void checkMasterLink(StorageDataSet dataSet, MasterLinkDescriptor masterLink)

Validates the *masterLink* property. When the *MasterLinkDescriptor's* *fetchAsNeeded* property is enabled (**true**), this method checks if there is a WHERE clause in the query. If no WHERE clause is specified, a *DataSetException* is thrown.

Overrides com.borland.dx.dataset.Provider.checkMasterLink
(com.borland.dx.dataset.StorageDataSet,
com.borland.dx.dataset.MasterLinkDescriptor)

executeStatement(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)

```
public static final int executeStatement(Database database, String statement, ReadWriteRow
    parameters)
```

Executes the SQL statement specified as *statement* and passes values to the parameter markers in the SQL statement (all named or unnamed) with values from the *ReadWriteRow*. Use this method to execute SQL statements with parameters that do not yield a *ResultSet*. For example:

```
QueryProvider.executeStatement(db1,"INSERT INTO CUST VALUES (?,?)", paramRow);
```

<i>database</i>	The <i>Database</i> object associated with the query.
<i>statement</i>	The query statement to execute.
<i>parameters</i>	The <i>ReadWriteRow</i> implementation that stores values for the query parameters.

For more information on other methods that execute statements, see “Executing queries” in the About section of the *QueryDescriptor* class.

executeStatement(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow[])

```
public static final int executeStatement(Database database, String statement, ReadWriteRow[]
    parameters)
```

Similar to `executeStatement(com.borland.dx.sql.dataset.Database, java.lang.String, com.borland.dx.dataset.ReadWriteRow)`, but allows for an array of query parameters.

<i>database</i>	The <i>Database</i> object associated with the query.
<i>statement</i>	The query statement to execute.
<i>parameters</i>	An array of <i>ReadWriteRow</i> objects that store values for the query parameters. For example,

```
QueryProvider.executeStatement(db1,"INSERT INTO CUST
    (FIRST_NAME, LAST_NAME) VALUES (:1.NAME, :2.NAME)", new
    ReadWriteRow[] {paramRow1, paramRow2});
```

Where tags “1.” and “2.” in front of the named parameter specifies which query parameter row to find the value from. If no tags are given, the parameter names are found by searching the parameter rows from left to right.

getQueryString(com.borland.dx.dataset.StorageDataSet)

```
public final String getQueryString(StorageDataSet sds)
```

Returns the query *String* associated with this *QueryDataSet*. This property is a short cut to the *queryString* property of the *QueryDescriptor* object.

QueryResolver component

dx.sql.dataset package

Extends com.borland.dx.sql.dataset.SQLResolver

Implements com.borland.dx.dataset.Designable, java.io.Serializable

The *QueryResolver* component is the default *Resolver* object of the *Database* component. It encapsulates *UpdateMode* behavior and uses SQL queries to resolve data changes to a server.

The *database* property of this component must be set to the *Database* component that this *QueryResolver* is associated with. Otherwise, a *DataSetException* is generated.

To make a query updatable:

- If the *updateMode* property is set to *ALL_COLUMNS* (the default), set the *tableName* property of the *QueryDataSet*
- If a *QueryResolver* component has been specified and the *updateMode* property is set to *KEY_COLUMNS*, set the *rowId* property on the columns used as an index to the table.
- If you have a BLOB column, set the searchable property to false. This indicates that this column should never be used in a WHERE clause of an update query. Most databases do not support this for BLOBs.

When troubleshooting resolution problems, columns other than those with edited values may be causing problems. For instance, the problem might stem from columns which are included in the WHERE clause of the update query.

If you get a message that no rows were affected by the update query, check whether someone meanwhile modified the original row. This would cause the WHERE clause to fail. Use the *DataSet.refetchRow()* method to get the original row from the server to determine if this is the case.

JDataStore automatically recognizes read-only data such as calculated columns, and attempts to discover other non-updatable columns. Some JDBC drivers seem unaware that such columns are read-only. For this reason, you should set column properties to not resolvable. While this seems redundant, this can prove crucial to your application.

Columns of certain data types and fields which are calculated on the server could cause the comparison in the WHERE clause of the update query to fail. Conditions which might cause this include:

- a *Column* is of imprecise data type (such as float or double).
- a *Column* contains String data that is of fixed length; some drivers do not pad strings with blanks, which leads to failures in comparison.
- a *Column* is calculated on the server, and successive calls to the *saveChanges* method are made without an intervening refresh.

Possible solutions to the above types of problems are:

- Set the *metaDataUpdate* property of the *QueryDataSet* to *MetaDataUpdate.NONE* and the *searchable* property of the column in question to false so that the column is not included in the WHERE clause of the update query. Note that you will have to set other properties as well to make the query updatable.
- Add a *QueryResolver* to your project and set the *updateMode* property of this component to *KEY_COLUMNS*.

If your server returns an error other than a constraint or integrity violation, check whether your driver supports prefixing field names with tables names, for example, *testtable.column1*. If not, set the *useTableName* property of the *Database* component to false.

In 1–1 relationships, the *QueryResolver* is able to resolve SQL queries that have more than one table reference. Metadata discovery detects which table each column belongs to and the default resolution order is set in the *resolveOrder* of the *StorageDataSet*. When using 1-Many relationships, use a separate master detail *DataSet*. For Many–1 relationships, use lookups.

The *resolverQueryTimeout* property can help in situations where an application is trying to read a locked row.

QueryResolver constructors

QueryResolver()

public QueryResolver()

Default constructor of a *QueryResolver* object.

QueryResolver properties

Property	Implemented in
class*	java.lang.Object
database	this class

Property	Implemented in
resolverQueryTimeout	this class
updateMode	this class

database

```
public Database getDatabase()
public void setDatabase(Database database)
```

The *DataBase* object associated with the *QueryResolver*. If not set, a *DataSetException* is generated.

resolverQueryTimeout

```
public final int getResolverQueryTimeout()
public final void setResolverQueryTimeout(int seconds)
```

Limits the number of seconds that the driver waits for a *Resolver's* insert/update/delete statement to execute. If the limit is exceeded, a *DataSetException* is thrown. An *SQLErrorException* is thrown if a database-access error occurs.

The *int* value in both setter and getter indicate the query time-out, in seconds. A value of zero indicates unlimited (no time-out).

updateMode

```
public int getUpdateMode()
public void setUpdateMode(int updateMode)
```

Determines how updated rows in the local *DataSet* are identified with the corresponding rows of the server table. Valid values for this property are defined in the *UpdateMode* class.

QueryResolver methods

Method	Implemented in
checkIfBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Resolver
clone()	java.lang.Object
close (com.borland.dx.dataset.StorageDataSet)	this class
closeStatements (com.borland.dx.dataset.StorageDataSet)	this class
deleteRow(com.borland.dx.dataset.DataSet)	this class
equals(java.lang.Object)	java.lang.Object
fetchResolverListener()	com.borland.dx.sql.dataset.SQLResolver

Method	Implemented in
finalize()	java.lang.Object
hashCode()	java.lang.Object
insertRow(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
resolveData (com.borland.dx.dataset.DataSet)	com.borland.dx.sql.dataset.SQLResolver
toString()	java.lang.Object
updateRow (com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

close(com.borland.dx.dataset.StorageDataSet)

public synchronized void close(StorageDataSet dataSet)

Frees any system resources such as *Databases*, statements, and so on that are associated with the specified *StorageDataSet*.

Overrides com.borland.dx.dataset.Resolver.close
(com.borland.dx.dataset.StorageDataSet)

closeStatements(com.borland.dx.dataset.StorageDataSet)

public void closeStatements(StorageDataSet dataSet)

Frees any system resources used for statements associated with the specified *StorageDataSet*.

Overrides com.borland.dx.sql.dataset.SQLResolver.closeStatements
(com.borland.dx.dataset.StorageDataSet)

deleteRow(com.borland.dx.dataset.DataSet)

public void deleteRow(DataSet dataSet)

Deletes the current row in the *DataSet* from the *Database*.

Overrides com.borland.dx.sql.dataset.SQLResolver.deleteRow
(com.borland.dx.dataset.DataSet)

insertRow(com.borland.dx.dataset.DataSet)

```
public void insertRow(DataSet dataSet)
```

Inserts the current row of the *DataSet* into the *Database* specified in this resolver's *database* property.

Overrides `com.borland.dx.sql.dataset.SQLResolver.insertRow`
(`com.borland.dx.dataset.DataSet`)

**updateRow(com.borland.dx.dataset.DataSet,
com.borland.dx.dataset.ReadWriteRow)**

```
public void updateRow(DataSet dataSet, ReadWriteRow oldDataRow)
```

Updates the current row of the specified *DataSet* in the *Database*.

Overrides `com.borland.dx.sql.dataset.SQLResolver.updateRow`
(`com.borland.dx.dataset.DataSet`, `com.borland.dx.dataset.ReadWriteRow`)

QueryResolver event listeners

This component is a source for the following event sets.

resolver

```
public synchronized void addResolverListener(ResolverListener listener)
public synchronized void removeResolverListener(ResolverListener listener)
```

ResolutionException class

dx.sql.dataset package

Extends `com.borland.dx.dataset.DataSetException`

Implements `com.borland.jb.util.ChainedException`, `java.io.Serializable`

The *ResolutionException* class defines error constants and behavior associated with error conditions encountered when resolving changes made to data back to its data source. It extends *DataSetException* and may include chained exceptions.

See also *DataSetException*

ResolutionException variables

Variable	Defined in
ALREADY_LOADING	com.borland.dx.dataset.DataSetException
BAD_PROCEDURE_PROPERTIES	com.borland.dx.dataset.DataSetException
BAD_QUERY_PROPERTIES	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN	com.borland.dx.dataset.DataSetException
CANNOT_CHANGE_COLUMN_DATA_TYPE	com.borland.dx.dataset.DataSetException
CANNOT_FIND_TABLE_NAME	com.borland.dx.dataset.DataSetException
CANNOT_IMPORT_NULL_DATASET	com.borland.dx.dataset.DataSetException
CANNOT_REFRESH	com.borland.dx.dataset.DataSetException
CANNOT_SAVE_CHANGES	com.borland.dx.dataset.DataSetException
CANNOT_UPDATE_SCOPED_DATA_ROW	com.borland.dx.dataset.DataSetException
CLASS_NOT_FOUND_ERROR	com.borland.dx.dataset.DataSetException
COLUMN_ALREADY_BOUND	com.borland.dx.dataset.DataSetException
COLUMN_NOT_IN_ROW	com.borland.dx.dataset.DataSetException
COLUMN_TYPE_CONFLICT	com.borland.dx.dataset.DataSetException
CONNECTION_DESCRIPTOR_NOT_SET	com.borland.dx.dataset.DataSetException
CONNECTION_NOT_CLOSED	com.borland.dx.dataset.DataSetException
DATA_FILE_LOAD_FAILED	com.borland.dx.dataset.DataSetException
dataSet	this class
DATASET_CORRUPT	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_ROWS	com.borland.dx.dataset.DataSetException
DATASET_HAS_NO_TABLES	com.borland.dx.dataset.DataSetException
DATASET_NOT_OPEN	com.borland.dx.dataset.DataSetException
DATASET_OPEN	com.borland.dx.dataset.DataSetException
DELETE_DUPLICATES	com.borland.dx.dataset.DataSetException
DELETE_FAILED	this class
DRIVER_NOT_LOADED_AT_RUNTIME	com.borland.dx.dataset.DataSetException
DRIVER_NOT_LOADED_IN_DESIGN	com.borland.dx.dataset.DataSetException
DUPLICATE_COLUMN_NAME	com.borland.dx.dataset.DataSetException
DUPLICATE_PRIMARY	com.borland.dx.dataset.DataSetException
EMPTY_COLUMN_NAMES	com.borland.dx.dataset.DataSetException
errorCode	com.borland.dx.dataset.DataSetException
EXCEPTION_CHAIN	com.borland.dx.dataset.DataSetException
exceptionChain	com.borland.dx.dataset.DataSetException
FIELD_POST_ERROR	com.borland.dx.dataset.DataSetException
GENERIC_ERROR	com.borland.dx.dataset.DataSetException
INCOMPATIBLE_DATA_ROW	com.borland.dx.dataset.DataSetException

Variable	Defined in
INSERT_FAILED	this class
INSUFFICIENT_ROWID	com.borland.dx.dataset.DataSetException
INVALID_AGG_DESCRIPTOR	com.borland.dx.dataset.DataSetException
INVALID_CLASS	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_POSITION	com.borland.dx.dataset.DataSetException
INVALID_COLUMN_TYPE	com.borland.dx.dataset.DataSetException
INVALID_DATA_FILE_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_FORMAT	com.borland.dx.dataset.DataSetException
INVALID_ITERATOR_USE	com.borland.dx.dataset.DataSetException
INVALID_SCHEMA_FILE	com.borland.dx.dataset.DataSetException
INVALID_SORT_COLUMN	com.borland.dx.dataset.DataSetException
INVALID_STORE_CLASS	com.borland.dx.dataset.DataSetException
INVALID_STORE_NAME	com.borland.dx.dataset.DataSetException
IO_ERROR	com.borland.dx.dataset.DataSetException
LINK_COLUMNS_ERROR	com.borland.dx.dataset.DataSetException
LINKFIELD_IN_USERPARAMETERS	com.borland.dx.dataset.DataSetException
LOADING_NOT_STARTED	com.borland.dx.dataset.DataSetException
MASTER_DETAIL_VIEW_ERROR	com.borland.dx.dataset.DataSetException
MASTER_NAVIGATION_ERROR	com.borland.dx.dataset.DataSetException
MISMATCH_PARAM_RESULT	com.borland.dx.dataset.DataSetException
MISMATCHED_PARAMETER_FORMAT	com.borland.dx.dataset.DataSetException
MISSING_MASTER_DATASET	com.borland.dx.dataset.DataSetException
MISSING_REPLACESTOREROW	com.borland.dx.dataset.DataSetException
MISSING_RESOLVER	com.borland.dx.dataset.DataSetException
MULTIPLE_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NEED_LOCATE_START_OPTION	com.borland.dx.dataset.DataSetException
NEED_PROCEDUREPROVIDER	com.borland.dx.dataset.DataSetException
NEED_QUERYPROVIDER	com.borland.dx.dataset.DataSetException
NEED_STORAGEDATASET	com.borland.dx.dataset.DataSetException
NEEDS_RECALC	com.borland.dx.dataset.DataSetException
NO_CALC_AGG_FIELDS	com.borland.dx.dataset.DataSetException
NO_CALC_FIELDS	com.borland.dx.dataset.DataSetException
NO_DATABASE_TO_RESOLVE	com.borland.dx.dataset.DataSetException
NO_NON_BLOB_COLUMNS	com.borland.dx.dataset.DataSetException
NO_PRIMARY_KEY	com.borland.dx.dataset.DataSetException
NO_PRIOR_ORIGINAL_ROW	com.borland.dx.dataset.DataSetException
NO_RESULT_SET	com.borland.dx.dataset.DataSetException
NO_ROWS_AFFECTED	com.borland.dx.dataset.DataSetException
NO_UPDATABLE_COLUMNS	com.borland.dx.dataset.DataSetException
NO_WHERE_CLAUSE	com.borland.dx.dataset.DataSetException
NON_EXISTENT_ROWID	com.borland.dx.dataset.DataSetException

Variable	Defined in
NOT_DATABASE_RESOLVER	com.borland.dx.dataset.DataSetException
NOT_SELECT_QUERY	com.borland.dx.dataset.DataSetException
NOT_UPDATEABLE	com.borland.dx.dataset.DataSetException
NULL_COLUMN_NAME	com.borland.dx.dataset.DataSetException
ONEPASS_INPUT_STREAM	com.borland.dx.dataset.DataSetException
PARAMETER_COUNT_MISMATCH	com.borland.dx.dataset.DataSetException
PARTIAL_SEARCH_FOR_STRING	com.borland.dx.dataset.DataSetException
PROCEDURE_FAILED	com.borland.dx.dataset.DataSetException
PROCEDURE_IN_PROCESS	com.borland.dx.dataset.DataSetException
PROVIDER_FAILED	com.borland.dx.dataset.DataSetException
PROVIDER_OWNED	com.borland.dx.dataset.DataSetException
QUERY_FAILED	com.borland.dx.dataset.DataSetException
QUERY_IN_PROCESS	com.borland.dx.dataset.DataSetException
READ_ONLY_STORE	com.borland.dx.dataset.DataSetException
REFRESHROW_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
REOPEN_FAILURE	com.borland.dx.dataset.DataSetException
RESOLVE_FAILED	this class
RESOLVE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
RESOLVE_PARTIAL	this class
RESTRUCTURE_IN_PROGRESS	com.borland.dx.dataset.DataSetException
SET_CALCULATED_FAILURE	com.borland.dx.dataset.DataSetException
SQL_ERROR	com.borland.dx.dataset.DataSetException
TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED	com.borland.dx.dataset.DataSetException
UNEXPECTED_END_OF_QUERY	com.borland.dx.dataset.DataSetException
UNKNOWN_COLUMN_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_DETAIL_NAME	com.borland.dx.dataset.DataSetException
UNKNOWN_PARAM_NAME	com.borland.dx.dataset.DataSetException
UNRECOGNIZED_DATA_TYPE	com.borland.dx.dataset.DataSetException
UPDATE_FAILED	this class
URL_NOT_FOUND	com.borland.dx.dataset.DataSetException
URL_NOT_FOUND_IN_DESIGN	com.borland.dx.dataset.DataSetException
WRONG_DATABASE	com.borland.dx.dataset.DataSetException

dataSet

public transient DataSet dataSet

The *DataSet* object that generated the *ResolutionException*.

DELETE_FAILED

```
public static final int DELETE_FAILED = BASE+2
```

Attempted to delete a value in a *DataSet* that has been set to *readOnly*.

INSERT_FAILED

```
public static final int INSERT_FAILED = BASE+1
```

Attempted to insert a value in a *DataSet* that has been set to *readOnly*.

RESOLVE_FAILED

```
public static final int RESOLVE_FAILED = BASE+4
```

If the *StorageDataSet.maxResolveErrors* property is set, a *ResolutionException* with this error code will be thrown when the maximum errors are encountered.

The resolution process is aborted; changes are rolled back.

See also *ResolveError*

RESOLVE_PARTIAL

```
public static final int RESOLVE_PARTIAL = BASE+5
```

If the *StorageDataSet.maxResolveErrors* property is set, a *ResolutionException* with this error code will be thrown when all rows have been processed.

Any successfully processed rows will be committed.

See also *ResolveError*

UPDATE_FAILED

```
public static final int UPDATE_FAILED = BASE+3
```

Master rows that have detail rows linked to them cannot be deleted or have their linking columns modified.

ResolutionException constructors

ResolutionException(int, com.borland.dx.dataset.DataSet, java.lang.String, java.lang.Exception)

```
public ResolutionException(int errorCode, DataSet dataSet, String message, Exception ex)
```

Constructs a *ResolutionException* object with the following parameters:

<i>errorCode</i>	The integer value associated with this error.
<i>dataSet</i>	The <i>DataSet</i> object that generated this error.

<i>message</i>	The string message associated with this error.
<i>ex</i>	The <i>Exception</i> object.

```
ResolutionException(int, com.borland.dx.dataset.StorageDataSet[],
java.lang.String)
```

```
public ResolutionException(int errorCode, StorageDataSet[] errorDataSets, String message)
```

Constructs a *ResolutionException* object with the following parameters:

<i>errorCode</i>	The integer value associated with this error.
<i>errorDataSets</i>	The array of <i>StorageDataSet</i> objects that generated this error.
<i>message</i>	The string message associated with this error.

ResolutionException properties

Property	Implemented in
class*	java.lang.Object
dataSet*	this class
errorCode*	com.borland.dx.dataset.DataSetException
errorDataSets*	this class
exceptionChain*	com.borland.dx.dataset.DataSetException
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

dataSet

```
public DataSet getDataSet()
```

Returns an array of *StorageDataSets* that contain errors from a resolution process. A resolution process can be invoked by calling *DataSet.saveChanges()* or *Database.saveChanges()*. This property will only be non-null if one or more *StorageDataSets* that participated in the resolver operation had the *StorageDataSet.maxResolveErrors* property set to -1 or to a value greater than 0.

The returned array has a *StorageDataSet* entry for each *StorageDataSet* that participated in the resolve operation. The order of these *errorDataSets* in the array corresponds directory to the order of the *StorageDataSets* in the array of *StorageDataSets* passed into the resolution manager class.

In the case that `Database.saveChanges()` is called to invoke the resolution manager, the array passed to this method dictates the order of the `errorDataSets` array if there are any errors logged. If there are no errors logged

for a *StorageDataSet* being resolved, its entry in the *errorDataSets* array will be null.

The structure of the *errorDataSets* is described in *ResolveError*.

errorDataSets

```
public StorageDataSet[] getErrorDataSets()
```

Read-only property that returns all the errors for which *ErrorResponse.ignore()* was called.

This method returns an array of *StorageDataSets*. There is one *StorageDataSet* for every *DataSet* that participated in the resolution. The order of the *StorageDataSets* in the error *StorageDataSet* array corresponds to the order of the *DataSets* that were passed into the *Database.saveChanges()* method.

ResolutionException methods

Method	Implemented in
<code>addExceptionListener</code> (<code>com.borland.dx.dataset.ExceptionListener</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>badProcedureProperties()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>badQueryProperties()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>classNotFoundException</code> (<code>java.lang.ClassNotFoundException</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>clone()</code>	<code>java.lang.Object</code>
<code>connectionDescriptorNotSet()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>connectionNotClosed</code> (<code>java.lang.Exception</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>dataSetHasNoTable()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>dataSetNotOpen()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>deleteDuplicates()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>driverNotLoadedAtRuntime</code> (<code>java.lang.String</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>driverNotLoadedInDesign</code> (<code>java.lang.String</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>equals</code> (<code>java.lang.Object</code>)	<code>java.lang.Object</code>
<code>fillInStackTrace()</code>	<code>java.lang.Throwable</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>getExceptionListeners()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>insufficientRowId()</code>	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidClass</code> (<code>java.lang.Class</code>)	<code>com.borland.dx.dataset.DataSetException</code>
<code>invalidClass</code> (<code>java.lang.String</code> , <code>java.lang.String</code>)	<code>com.borland.dx.dataset.DataSetException</code>

Method	Implemented in
invalidColumnType (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
invalidSQLType(int)	com.borland.dx.dataset.DataSetException
invalidStoreName(java.lang.String)	com.borland.dx.dataset.DataSetException
IOException(java.io.IOException)	com.borland.dx.dataset.DataSetException
mismatchedParameterFormat()	com.borland.dx.dataset.DataSetException
mismatchParamResult()	com.borland.dx.dataset.DataSetException
missingMasterDataSet()	com.borland.dx.dataset.DataSetException
mkUrlNotFound(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
mkUrlNotFoundInDesign(java.lang.String, java.lang.Exception)	com.borland.dx.dataset.DataSetException
multipleRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
needProcedureProvider()	com.borland.dx.dataset.DataSetException
needQueryProvider()	com.borland.dx.dataset.DataSetException
needsRecalc(java.lang.String)	com.borland.dx.dataset.DataSetException
noDatabaseOnResolver()	com.borland.dx.dataset.DataSetException
nonExistentRowId()	com.borland.dx.dataset.DataSetException
noResultSet()	com.borland.dx.dataset.DataSetException
noRowsAffected(java.lang.String)	com.borland.dx.dataset.DataSetException
notDatabaseResolver()	com.borland.dx.dataset.DataSetException
notify()	java.lang.Object
notifyAll()	java.lang.Object
notSelectQuery()	com.borland.dx.dataset.DataSetException
notSortable()	com.borland.dx.dataset.DataSetException
noUpdatableColumns()	com.borland.dx.dataset.DataSetException
noWhereClause (com.borland.dx.dataset.DataSet)	com.borland.dx.dataset.DataSetException
onePassInputStream (com.borland.dx.dataset.Column)	com.borland.dx.dataset.DataSetException
parameterCountMismatch(int, int, int)	com.borland.dx.dataset.DataSetException
printStackTrace()	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintStream)	com.borland.dx.dataset.DataSetException
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
procedureFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
procedureInProgress()	com.borland.dx.dataset.DataSetException
providerFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
providerOwned()	com.borland.dx.dataset.DataSetException
queryFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
queryInProgress()	com.borland.dx.dataset.DataSetException
readOnlyStore(java.lang.String)	com.borland.dx.dataset.DataSetException

Method	Implemented in
removeExceptionListener (com.borland.dx.dataset. ExceptionListener)	com.borland.dx.dataset.DataSetException
resolveFailed(java.lang.Exception)	com.borland.dx.dataset.DataSetException
SQLException(java.sql.SQLException)	com.borland.dx.dataset.DataSetException
throwException(int, java.lang.Exception)	com.borland.dx.dataset.DataSetException
throwExceptionChain (java.lang.Throwable)	com.borland.dx.dataset.DataSetException
toString()	java.lang.Throwable
transactionIsolationLevelNotSupported()	com.borland.dx.dataset.DataSetException
unexpectedEndOfQuery()	com.borland.dx.dataset.DataSetException
unknownColumnName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownDetailName(java.lang.String)	com.borland.dx.dataset.DataSetException
unknownParamName(java.lang.String)	com.borland.dx.dataset.DataSetException
unrecognizedDataType()	com.borland.dx.dataset.DataSetException
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
wrongDatabase()	com.borland.dx.dataset.DataSetException

ResolveError class

dx.sql.dataset package

Extends java.lang.Object

Implements java.io.Serializable

This class provides information about a specific row that could not be resolved.

The *ResolutionManager* that is invoked by calling *DataSet.saveChanges()* or *Database.saveChanges()* will log errors in an error *DataSet* for all rows that meet the following criteria:

- One or more *StorageDataSets* being resolved has the *StorageDataSet.maxResolveErrors* property set to a value greater than 0. If this property is set to -1, all errors will be logged.
- An exception (i.e. *SQLException*) was encountered for the row as it was being inserted, deleted or updated.

At the end of the resolution process that is not aborted, a *ResolutionException* with an error code of *ResolutionException.RESOLVE_PARTIAL* will be thrown after an attempt to resolve all rows is made and the successfully resolved rows are committed. If the *StorageDataSet.MaxResolveErrors* limit is reached, a

ResolutionException with an error code of *ResolutionException.RESOLVE_FAILED* will be thrown.

The *ResolutionException.getErrorDataSets()* method can be used to get all the errors for which *ErrorResponse.ignore()* was called. This method returns an array of *StorageDataSets*. There is one *StorageDataSet* for every *DataSet* that participated in the resolution. The order of the *StorageDataSets* in the error *StorageDataSet* array corresponds to the order of the *DataSets* that were passed into the *Database.saveChanges()* method.

If there are multiple *StorageDataSets* being resolved and a particular *DataSet* did not encounter any errors, its index in the error *StorageDataSet* array will be null.

The error *StorageDataSet* array has the same columns as its corresponding *DataSet* that was resolved plus one extra *RESOLVE_ERROR* column. The *RESOLVE_ERROR* column is of type *Variant.OBJECT* and is appended as the last column in the error *StorageDataSet*. The *RESOLVE_ERROR* column contains an instance of the *ResolveError* class.

See also *StorageDataSet.maxResolveErrors*, *ResolutionException.RESOLVE_PARTIAL*, *ResolutionException.RESOLVE_FAILED*

ResolveError variables

Variable	Defined in
category	this class
code	this class
context	this class
ex	this class
internalRow	this class
message	this class
response	this class
row	this class

category

public int category

Currently not used for JDBC related errors.

code

public int code

Currently set to *ex.getErrorCode()* if the exception is extended from *DataSetException*.

If the error was caused by a *java.sql.SQLException*, this is the **int** value returned from calling *SQLException.getErrorCode()*.

context

public String context

If the error is caused by a *java.sql.SQLException*, this is the *String* value returned from calling *SQLException.getMessage()*, concatenated with the return value of *SQLException.getSQLState*.

ex

public Exception ex

If not **null**, the *Exception* that caused this error.

internalRow

public long internalRow

DataSet *internalRow* that encountered the error. *internalRow* is a unique identifier for the row.

message

public String message

The error message for the error. Usually the same as *ex.getMessage()*.

response

public int response

Set to *ErrorResponse.ABORT* if the error terminated the resolution transaction.

Set to *ErrorResponse.IGNORE* if the error was ignored.

row

public int row

DataSet *row* that encountered the error. In the case of detail *DataSets*, this is not always unique since a single detail *DataSet* can have multiple groups of records with identical row values.

ResolveError properties

Property	Implemented in
class*	java.lang.Object

ResolveError methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

SQLDialect interface

dx.sql.dataset package

The *SQLDialect* interface defines constants for SQL database servers.

SQLDialect variables

Variable	Defined in
INTERBASE	this class
ORACLE	this class
UNKNOWN	this class

INTERBASE

public static final int INTERBASE = 0x2

An InterBase server.

ORACLE

public static final int ORACLE = 0x3

An Oracle server.

UNKNOWN

public static final int UNKNOWN = 0x1

A server of an unknown type.

SQLResolutionManager component

dx.sql.dataset package

Extends com.borland.dx.sql.dataset.ResolutionManager

Implements com.borland.dx.sql.dataset.DefaultResolver,
com.borland.dx.sql.dataset.TransactionSupport

The *SQLResolutionManager* component performs most of the work for the resolution process. The algorithms for transaction management and the change resolution reside in this object. An instance of this class is instantiated with an implementation of the *DefaultResolver* and *TransactionSupport* interfaces.

If a **null** *TransactionSupport* object is passed in, the *SQLResolutionManager* takes no action for transaction processing (as is applicable to non-transaction-processing data sources). This object manages the resolution process as follows:

- The array of *DataSet* components passed into the *saveChanges()* method is analyzed and split into a list of trees. Each tree encapsulates any one-to-many-to-many (and so on) behavior.
- A non-linked (standalone) *DataSet* is represented as a tree with no children.

The resolution process is broken into two different algorithms. For stand-alone *DataSet* components, all rows in the following categories are processed in this order to preserve the integrity of the data:

- *Deleted* rows
- *Modified* rows
- *Inserted* rows

For one-to-many *DataSet* relationships, all rows in the following categories are processed in this order:

- *Deleted* rows for the bottom-most children *DataSet* components
- parents of the above *DataSet* components are recursively processed:
 - *Inserted* rows, starting at the root *DataSet* and working recursively downward
 - *Modified* rows, bottom-up.

Since changes are not in sequential order, changing link fields in a one-to-many relationship can cause data loss. Therefore, this action is disallowed by default.

SQLResolutionManager constructors

SQLResolutionManager()

public SQLResolutionManager()

Constructs a *SQLResolutionManager* object.

SQLResolutionManager properties

Property	Implemented in
class*	java.lang.Object
database	this class
doTransactions	this class
insertsBeforeUpdates	com.borland.dx.sql.dataset.ResolutionManager
postEdits	com.borland.dx.sql.dataset.ResolutionManager
resetPendingStatus	com.borland.dx.sql.dataset.ResolutionManager
transactionSupport**	this class

database

public Database getDatabase()

public void setDatabase(Database database)

Specifies the *Database* object that this component is associated with.

doTransactions

public boolean isDoTransactions()

public void setDoTransactions(boolean doTransactions)

Specifies whether transactions are supported or not.

transactionSupport

public void setTransactionSupport(TransactionSupport transactionSupport)

Write-only property that overwrites the superclass setter to make this class the default *TransactionSupport* object.

SQLResolutionManager methods

Method	Implemented in
clone()	java.lang.Object
commit()	this class

Method	Implemented in
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>getResolver</code> (<code>com.borland.dx.dataset.DataSet</code>)	this class
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>initError(int,</code> <code>com.borland.dx.dataset.DataSet,</code> <code>com.borland.dx.dataset.DataSet,</code> <code>resolveError)</code>	this class
<code>notify()</code>	<code>java.lang.Object</code>
<code>notifyAll()</code>	<code>java.lang.Object</code>
<code>resetPendingStatus</code> (<code>com.borland.dx.dataset.DataSet[]</code> , <code>boolean</code>)	<code>com.borland.dx.sql.dataset.ResolutionManager</code>
<code>rollback()</code>	this class
<code>saveChanges</code> (<code>com.borland.dx.dataset.DataSet</code>)	<code>com.borland.dx.sql.dataset.ResolutionManager</code>
<code>saveChanges</code> (<code>com.borland.dx.dataset.DataSet[]</code>)	<code>com.borland.dx.sql.dataset.ResolutionManager</code>
<code>setDefaultResolver</code> (<code>com.borland.dx.sql.dataset.</code> <code>DefaultResolver</code>)	<code>com.borland.dx.sql.dataset.ResolutionManager</code>
<code>setTransactionSupport</code> (<code>com.borland.dx.sql.dataset.</code> <code>TransactionSupport</code>)	<code>com.borland.dx.sql.dataset.ResolutionManager</code>
<code>start()</code>	this class
<code>toString()</code>	<code>java.lang.Object</code>
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>

`initError(int, com.borland.dx.dataset.DataSet,` `com.borland.dx.dataset.DataSet, resolveError)`

protected final void `initError(int code, DataSet dataSet, DataSet view, ResolveError resolveError)`

Adds extra error context information to *resolveError* object.

code One of `UPDATE_FAILED`, `INSERT_FAILED`, or `DELETE_FAILED`, as specified in the *ResolutionException* variables.

dataSet The base *DataSet* passed into the *ResolutionManager*.

	<i>view</i>	Positioned at the row that caused the error.
	<i>ResolveError</i>	Can be used to record additional error information. See <i>ResolveError</i> for more information.
Overrides	com.borland.dx.sql.dataset.ResolutionManager.initError(int, com.borland.dx.sql.dataset.DataSet, com.borland.dx.sql.dataset.DataSet, resolveError)	

SQLResolver class (abstract)

dx.sql.dataset package

Extends	com.borland.dx.sql.dataset.Resolver
Extended by	com.borland.dx.sql.dataset.ProcedureResolver, com.borland.dx.sql.dataset.QueryResolver
Implements	com.borland.dx.sql.dataset.Designable, java.io.Serializable
	This class allows for alternate implementations of the actual behavior required to save changes made to a <i>QueryDataSet</i> for example, to its database data source.
	The <i>SQLResolutionManager</i> requires a <i>Resolver</i> that extends this class. The <i>saveChanges()</i> method of a <i>QueryDataSet</i> and <i>ProcedureDataSet</i> instantiate a <i>SQLResolutionManager</i> . The <i>Resolver</i> of a <i>QueryDataSet</i> should therefore be an instance of a class that extends <i>SQLResolver</i> , such as the <i>QueryResolver</i> .

SQLResolver properties

Property	Implemented in
class*	java.lang.Object
database	this class

database

public abstract Database getDatabase()
public abstract void setDatabase(Database database)

Specifies the *Database* that is the target of the data changes.

SQLResolver methods

Method	Implemented in
checkIfBusy (com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Resolver
clone()	java.lang.Object
close(com.borland.dx.dataset.StorageDataSet)	com.borland.dx.dataset.Resolver
closeStatements (com.borland.dx.dataset.StorageDataSet)	this class
deleteRow(com.borland.dx.dataset.DataSet)	this class
equals(java.lang.Object)	java.lang.Object
fetchResolverListener()	this class
finalize()	java.lang.Object
hashCode()	java.lang.Object
insertRow(com.borland.dx.dataset.DataSet)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
resolveData(com.borland.dx.dataset.DataSet)	this class
toString()	java.lang.Object
updateRow(com.borland.dx.dataset.DataSet, com.borland.dx.dataset.ReadWriteRow)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

closeStatements(com.borland.dx.dataset.StorageDataSet)

public abstract void closeStatements(StorageDataSet dataSet)

Closes any open statements cached by a *Resolver*.

deleteRow(com.borland.dx.dataset.DataSet)

public abstract void deleteRow(DataSet dataSet)

Instructs the *Resolver* to delete the current row in the *DataSet* from the *Database*.

fetchResolverListener()

public ResolverListener fetchResolverListener()

A *Resolver* can optionally have a *resolverListener* property. This method allows the *ResolutionManager* to get the listener. It then issues events before and after each change to the *Database*.

insertRow(com.borland.dx.dataset.DataSet)

public abstract void insertRow(DataSet dataSet)

Instructs the *Resolver* to insert the current row of the *DataSet* into the *Database*.

resolveData(com.borland.dx.dataset.DataSet)

public void resolveData(DataSet dataSet)

Calls the *saveChanges()* method on the current *Database*.

Overrides com.borland.dx.dataset.Resolver.resolveData
(com.borland.dx.dataset.DataSet)

**updateRow(com.borland.dx.dataset.DataSet,
com.borland.dx.dataset.ReadWriteRow)**

public abstract void updateRow(DataSet dataSet, ReadWriteRow oldRow)

Instructs the *Resolver* to update the current row of the *DataSet* in the *Database*.

SQLResolver event listeners

This class is a source for the following event sets.

resolver

public synchronized void addResolverListener(ResolverListener listener)

public synchronized void removeResolverListener(ResolverListener listener)

dx.text package

The *com.borland.dx.text* package contains classes and interfaces that:

- Control alignment and formatting of objects
- Control formatting of data and values
- Handle formatting and parsing exceptions
- Handle input validation

The *dx.text* package contains the following types of classes:

- ItemEditor classes
- ItemFormatter classes
- Exception-related classes
- Placement-related classes

The following classes and interfaces in this package are used internally by classes in this and other *com.borland* packages. These classes and interfaces are not intended for general use and are not documented. Do not use these classes and interfaces directly in your application.

- ItemEditMaskChar
- ItemEditMaskRegion
- ItemEditMaskRegionChar

For more information, visit the database newsgroup. Details on newsgroups can be found at <http://www.borland.com/newsgroups>. The database newsgroup is dedicated to issues about writing database applications and is actively monitored by our support engineers as well as the Development team.

Interfaces

ItemEditMask

ItemEditMaskChar

ItemEditMaskRegionChar

Classes and components

Alignment	BigDecimalFormatter	BinaryFormatter
BooleanFormat	BooleanFormatter	ByteFormatter
DateFormatter	DoubleFormatter	IntegerFormatter
InvalidFormatException	ItemEditMaskRegion	ItemEditMaskState
ItemEditMaskStr	ItemFormatStr	ItemFormatter
LongFormatter	ObjectFormatter	ShortFormatter
SimpleFormatter	StringFormatter	TextFormat
TimeFormatter	TimestampFormatter	VariantFormatStr
VariantFormatter		

Overview of classes in the *com.borland.dx.text* package

ItemEditor classes

An Item Editor class involves character-by-character control of input.

ItemEditMask	An open interface for character-by-character input validation.
ItemEditMaskState	Carries state information for a control while the control is using an ItemEditMask interface. The control owns this information, though it is instantiated by the <i>ItemEditMask</i> . This allows multiple controls to share a common <i>ItemEditMask</i> .
ItemEditMaskStr	Implements the <i>ItemEditMask</i> interface using pattern strings to control formatting, parsing, and edit interactions.

ItemFormatter classes

An Item Formatter class translates a complete input value to or from a specified data type.

VariantFormatter	Base class to format and parse numeric data, time and date data, and strings. Implemented by <i>ItemFormatStr</i> , which uses JDK pattern strings. Supports numeric, currency, date-time, and text patterns.
VariantFormatStr	Extends the <i>VariantFormatter</i> class. Uses string patterns to control formatting and parsing. Supports numeric, currency, date-time, and text patterns.
ItemFormatStr	A simple wrapper class for <i>VariantFormatStr</i> . It was written primarily so there was a version of <i>VariantFormatStr</i> which implemented the basic <i>ItemFormatter</i> interface. This was done because the model-view architecture does not know about <i>variants</i> . This provides a way for the model-view classes to use <i>VariantFormatStr</i> for dataset activities without knowing about <i>variants</i> .
SimpleFormatter	A simple wrapper class around <i>VariantFormatter</i> which hides the details of the formatting pattern and the desired <i>Locale</i> . You just tell it the type of data you want to format or parse, and it uses <i>VariantFormatStr</i> , which handles all the processing. Primarily intended as an example.
ItemFormatter	An interface for translating a data object to and from a string.
BigDecimalFormatter	Implements <i>ItemFormatter</i> for the <i>BigDecimal</i> data type.
BooleanFormat	A string-based pattern used to create a <i>BooleanFormatter</i> object. Enables working with two values, stored as true or false , but formatted using string values you specify.
BinaryFormatter	Implements <i>ItemFormatter</i> for the <i>Binary</i> data type.
BooleanFormatter	Implements <i>ItemFormatter</i> for the boolean data type.
ByteFormatter	Used for formatting and parsing byte columns in <i>DataSets</i> . The range is -128 to 127.
DateFormatter	Implements <i>ItemFormatter</i> for the <i>Date</i> data type.
DoubleFormatter	Implements <i>ItemFormatter</i> for the double data type.

IntegerFormatter	Implements <i>ItemFormatter</i> for the int data type.
LongFormatter	Implements <i>ItemFormatter</i> for the long data type.
ObjectFormatter	Implements <i>ItemFormatter</i> for the Java Object data type.
ShortFormatter	Like <i>ByteFormatter</i> , but for short columns (-32767 to 32768).
StringFormatter	Implements <i>ItemFormatter</i> for the String data type.
TextFormat	Extends the basic <i>Format</i> class but allows for special formatting of text. Uses standard control patterns for fill characters and replace characters.
TimeFormatter	Implements <i>ItemFormatter</i> for the Time data type.
TimestampFormatter	Implements <i>ItemFormatter</i> for the Timestamp data type.

Exception-related classes

InvalidFormatException	The <i>Exception</i> class generated by format and parser code exceptions.
------------------------	--

Placement-related classes

Alignment	Provides general-purpose two-dimensional alignment constants for aligning an object within a rectangular container.
-----------	---

Alignment class

dx.text package

Extends

java.lang.Object

The *Alignment* component provides general-purpose two-dimensional alignment constants for aligning an object within a rectangular container. The constants are divided into the following groups. You can select none or one (the maximum) per group:

Horizontal alignment:

- LEFT
- CENTER
- RIGHT
- HSTRETCH
- HORIZONTAL

Vertical alignment:

- TOP
- MIDDLE
- BOTTOM
- VSTRETCH
- VERTICAL

Alignment variables

Variable	Defined in
BOTTOM	this class
CENTER	this class
HORIZONTAL	this class
HSTRETCH	this class
LEFT	this class
MIDDLE	this class
RIGHT	this class
TOP	this class
UNDEFINED	this class
VERTICAL	this class
VSTRETCH	this class

BOTTOM

```
public static final int BOTTOM = 0x30
```

A vertical alignment constant. Aligns vertically along the bottom of the container.

CENTER

```
public static final int CENTER = 0x02
```

A horizontal alignment constant. Centers horizontally within the container object.

HORIZONTAL

```
public static final int HORIZONTAL = 0x0F
```

Used to programmatically filter out the set of vertical alignment bits, leaving only the horizontal alignment.

HSTRETCH

```
public static final int HSTRETCH = 0x04
```

A horizontal alignment constant. Aligns horizontally between the left and right edges of the container, stretching as necessary.

LEFT

```
public static final int LEFT = 0x01
```

A horizontal alignment constant. Aligns to the horizontal left of the container.

MIDDLE

```
public static final int MIDDLE = 0x20
```

A vertical alignment constant. Aligns in the vertical middle of the container.

RIGHT

```
public static final int RIGHT = 0x03
```

A horizontal alignment constant. Aligns horizontally along the right edge of the container.

TOP

```
public static final int TOP = 0x10
```

A vertical alignment constant. Aligns along the top of the container.

UNDEFINED

```
public static final int UNDEFINED = 0x00
```

No alignment constant is defined.

VERTICAL

```
public static final int VERTICAL = 0xF0
```

Used to programmatically filter out the set of horizontal alignment bits, leaving only the vertical alignment.

VSTRETCH

```
public static final int VSTRETCH = 0x40
```

A vertical alignment constant. Aligns vertically between the top and bottom of the container, stretching if necessary.

Alignment properties

Property	Implemented in
class*	java.lang.Object

Alignment methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
valid(int)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

valid(int)

public static final boolean valid(int alignment)

Returns **true** if the specified value is a valid alignment value. Otherwise, this method returns **false**.

alignment Any valid *Alignment* variable.

BigDecimalFormatter class

dx.text package

Extends	com.borland.dx.text.VariantFormatter
Implements	java.io.Serializable
	The <i>BigDecimalFormatter</i> class formats and parses the <i>java.math.BigDecimal</i> values.
See also	String-based patterns (masks)

BigDecimalFormatter constructors

BigDecimalFormatter(int)

public BigDecimalFormatter(int scale)

Constructs a *BigDecimalFormatter* object.

scale Specify *scale* as a power of 10. The integer value is divided by implicitly.

BigDecimalFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type, which is always *Variant.BIGDECIMAL* for *BigDecimalFormatter*.

BigDecimalFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object

Method	Implemented in
parse(com.borland.dx.dataset.Variant, char[], int, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public String format(Variant value)

Returns a *String* representing the given *BigDecimal* value stored in the supplied *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(java.lang.String, com.borland.dx.dataset.Variant)

public void parse(String stringValue, Variant value)

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue The string to be parsed.

value The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

BinaryFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *BinaryFormatter* component is the default formatter and parser of *Variant.INPUTSTREAM* type data. This is a placeholder class that ensures that formatting requests of binary values do not generate an *Exception*.

See also String-based patterns (masks)

BinaryFormatter constructors

BinaryFormatter()

public BinaryFormatter()

Constructs a *BinaryFormatter* object.

BinaryFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class
locale*	this class
pattern*	this class
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

formatObj

public Format getFormatObj()

This property is used internally by other *com.borland* classes. You should never use this property directly.

locale

public Locale getLocale()

This property is used internally by other *com.borland* classes. You should never use this property directly.

pattern

```
public String getPattern()
```

This property is used internally by other *com.borland* classes. You should never use this property directly.

variantType

```
public int getVariantType()
```

This property is used internally by other *com.borland* classes. You should never use this property directly.

BinaryFormatter methods

Method	Implemented in
<code>clone()</code>	<code>java.lang.Object</code>
<code>equals(java.lang.Object)</code>	<code>java.lang.Object</code>
<code>finalize()</code>	<code>java.lang.Object</code>
<code>format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>format(com.borland.dx.dataset.Variant)</code>	this class
<code>format(java.lang.Object)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>getSpecialObject(int)</code>	this class
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>notify()</code>	<code>java.lang.Object</code>
<code>notifyAll()</code>	<code>java.lang.Object</code>
<code>parse(com.borland.dx.dataset.Variant, char[], int, int)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>parse(java.lang.String, com.borland.dx.dataset.Variant, int)</code>	this class
<code>parse(java.lang.String, com.borland.dx.dataset.Variant)</code>	this class
<code>parse(java.lang.String)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>setFromDouble (com.borland.dx.dataset.Variant, int, double)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>setFromInt(com.borland.dx.dataset.Variant, int, int)</code>	<code>com.borland.dx.text.VariantFormatter</code>
<code>setPattern(java.lang.String)</code>	this class
<code>setSpecialObject(int, java.lang.Object)</code>	this class
<code>toString()</code>	<code>java.lang.Object</code>
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>

format(com.borland.dx.dataset.Variant)

```
public String format(Variant value)
```

Returns a *String* representing the given value stored in the supplied object. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

<i>value</i>	The value to be formatted to a <i>String</i> .
--------------	--

Overrides `com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)`

getSpecialObject(int)

```
public Object getSpecialObject(int objType)
```

Returns the value of the specified special object.

Some formatter classes define special objects for their own use. You must know the internal details of the Format subclass being used to use *getSpecialObject()*.

<i>objType</i>	The special object type to return.
----------------	------------------------------------

Overrides `com.borland.dx.text.VariantFormatter.getSpecialObject(int)`

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output an *Object* containing the appropriate value. A **null** return value results when *stringValue* is **null** or empty.

<i>stringValue</i>	The string to be parsed.
--------------------	--------------------------

```
Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String,  
com.borland.dx.dataset.Variant)
```

```
parse(java.lang.String, com.borland.dx.dataset.Variant, int)
```

```
public void parse(String stringValue, Variant value, int variantType)
```

An alternative form of *parse()* that allows the type of *Variant* returned to be specified.

<i>stringValue</i>	The string to be parsed. A null or empty <i>stringValue</i> returns a <i>VariantAssignedNull</i> variant.
--------------------	--

value The *Variant* that receives the resulting data.

variantType The desired type of variant. If *variantType* is zero or one of the *VariantIsNull* types, the method chooses the default variant type specified at the time of the construction of *VariantFormatter*.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant, int)

setPattern(java.lang.String)

public String setPattern(String pattern)

Sets the pattern used for parsing and formatting to a new pattern, returning the old pattern. The new pattern must be of the same basic type associated with this type of formatter. For example, if you used a Date/Time pattern in the constructor, you can't switch to a numeric pattern as each basic pattern type has its own data-dependent *format()* and *parse()* methods.

If the new pattern is **null** (or empty), *setPattern()* chooses a default pattern for the current locale.

pattern The new pattern to be used for formatting and parsing.

Overrides com.borland.dx.text.VariantFormatter.setPattern(java.lang.String)

setSpecialObject(int, java.lang.Object)

public Object setSpecialObject(int objType, Object obj)

Some formatter classes define special objects for their own use. This method allows them to be set. You must know the internal details of the Format subclass being used to use *setSpecialObject()*.

The returned value is the prior value of the object.

objType The special object type. The possible values are *VariantFormatter.FillChar*, which is the fill character to fill blank slots, and *VariantFormatter.Replacecharacter*, which is used to replace *FillChar* on parse.

obj The special object.

Overrides com.borland.dx.text.VariantFormatter.setSpecialObject(int, java.lang.Object)

BooleanFormat component

dx.text package

Extends java.text.Format

Implements java.io.Serializable, java.lang.Cloneable

The *BooleanFormat* component uses a string-based pattern, typically a *java.text.Format* used to control the formatting of boolean values. This class is helpful when working with values that can have two values, stored as **true** or **false**, but formatted using string values you specify. For example, you can store gender information in a column of type **boolean** but have *JDataStore* format the field to display and accept input values of “Male” and “Female”. In addition, the *BooleanFormat* class allows a third string to display for **null** values for data that has not yet been entered.

The *BooleanFormat* pattern consists of three parts, separated by semicolons:

- The format string for **true** values
- The format string for **false** values
- The format string for **null** values, for example, when a field is left blank. If this part of the pattern is not supplied, *JDataStore* stores the default value of **false** for blank data values and formats the data according to the format string for **false** values.

The following table illustrates valid patterns and their formatting effects:

Table 6.1 BooleanFormat patterns and formats

BooleanFormat specification	Format for true values	Format for false values	Format for null values
"T;F;F"	T	F	F
"male;female"	Male	female	female
"smoker;;"	smoker	(blank)	(blank)
"smoker;nonsmoker;(unknown)"	smoker	nonsmoker	(unknown)

This class can be assigned to the following *Column* properties:

- *displayMask*
- *editMask*
- *exportDisplayMask*

When used as an *editMask*, the field width for data entry is set to the longest of the three parts in the pattern. All characters of the pattern are set as optional which means that the user is able to type any values into this field when entering or editing data. When leaving the field, the formatter is used to validate the data entered against the pattern specification and if invalid, will generate a *ValidationException*.

You can input abbreviations if your pattern is specified such that each part is a unique string. For example, the following table illustrates the values entered for various input into a field with a specification of “Yes;No;Don’t know”:

Input value	Parsed and stored as
Yes	true
y	true
No	false
Don’t know	assigned null
Y	true
Ye	true
N	false
D	assigned null
"" (empty string)	assigned null

The following table illustrates the parser logic where a pattern specification contains similar initial characters, and whether the parser can or cannot determine values based on abbreviated input. For a format specification of “marmot;monkey;mongoose”, the following input values yield results of:

Input value	Parsed and stored as
M	(ambiguous - cannot parse)
mon	(ambiguous - cannot parse)
ma	marmot
Mong	mongoose

Note JDataStore ignores upper and lowercase differences when parsing data against a pattern specification.

BooleanFormat constructors

BooleanFormat()

```
public BooleanFormat()
```

Creates a *BooleanFormat* object.

BooleanFormat(java.lang.String)

```
public BooleanFormat(String pattern)
```

Creates a *BooleanFormat* object with the specified pattern.

pattern The *String* pattern that specifies formatting of **true**, **false**, and **null** values.

BooleanFormat properties

Property	Implemented in
class*	java.lang.Object
falseString*	this class
nullString*	this class
trueString*	this class

falseString

public String getFalseString()

Read-only property that returns the part of the *BooleanFormat* specification which deals with the format of **false** values.

To set this property, use the *BooleanFormat(String)* constructor or the *applyPattern()* method.

nullString

public String getNullString()

Read-only property that returns the part of the *BooleanFormat* specification which deals with the format of **null** values.

To set this property, use the *BooleanFormat(String)* constructor or the *applyPattern()* method.

trueString

public String getTrueString()

Read-only property that returns the part of the *BooleanFormat* specification which deals with the format of **true** values.

To set this property, use the *BooleanFormat(String)* constructor or the *applyPattern()* method.

BooleanFormat methods

Method	Implemented in
applyPattern(java.lang.String)	this class
clone()	java.text.Format
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object

Method	Implemented in
<code>format(java.lang.Boolean, java.lang.StringBuffer, java.text.FieldPosition)</code>	this class
<code>format(java.lang.Object, java.lang.StringBuffer, java.text.FieldPosition)</code>	this class
<code>format(java.lang.Object)</code>	<code>java.text.Format</code>
<code>hashCode()</code>	<code>java.lang.Object</code>
<code>notify()</code>	<code>java.lang.Object</code>
<code>notifyAll()</code>	<code>java.lang.Object</code>
<code>parse(java.lang.String, java.text.ParsePosition)</code>	this class
<code>parseObject(java.lang.String, java.text.ParsePosition)</code>	this class
<code>parseObject(java.lang.String)</code>	<code>java.text.Format</code>
<code>toPattern()</code>	this class
<code>toString()</code>	<code>java.lang.Object</code>
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>

applyPattern(java.lang.String)

`public void applyPattern(String pattern)`

Sets the *BooleanFormat* specification that defines the format of **true**, **false**, and **null** values. If an empty pattern is used, this method defaults to “**true>false**”.

pattern The format specification.

format(java.lang.Boolean, java.lang.StringBuffer, java.text.FieldPosition)

`public StringBuffer format(Boolean value, StringBuffer result, FieldPosition pos)`

This method formats a boolean value into a *java.lang.StringBuffer*.

value The **boolean** value to format. Defaults to **null**.

result Where the formatted string should be appended to.

pos On input: an alignment field, if desired. On output: the offsets of the alignment field.

format(java.lang.Object, java.lang.StringBuffer, java.text.FieldPosition)

```
public final StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos)
```

This method formats a boolean value into a *java.lang.StringBuffer*. For more information on this method, see *java.text.Format.format(Object, StringBuffer, FieldPosition)*.

obj The *Object* to format. Defaults to **null**.
toAppendTo Where the formatted string should be appended to.
pos On input: an alignment field, if desired. On output: the offsets of the alignment field.

Overrides `java.text.Format.format(Object, StringBuffer, FieldPosition)`

parse(java.lang.String, java.text.ParsePosition)

```
public Boolean parse(String text, ParsePosition pos)
```

This method parses a string into a boolean value. For more information on this method, see *java.text.Format.parseObject(String, ParsePosition)*.

text The string containing a value to be parsed.
pos On input: an alignment field, if desired. On output: the offsets of the alignment field.

parseObject(java.lang.String, java.text.ParsePosition)

```
public final Object parseObject(String source, ParsePosition pos)
```

This method parses a string into an *Object*. A return value of **null** equals a blank string or the third part of the *BooleanFormat* pattern. An error in parsing is indicated by returning a **null** and by not advancing the parse position index. For more information on this method, see *java.text.Format.parseObject(String, ParsePosition)*.

source The string containing a value to be parsed.
pos On input: an alignment field, if desired. On output: the offsets of the alignment field.

Overrides `java.text.Format.parseObject(String, ParsePosition)`

toPattern()

```
public String toPattern()
```

Returns the *String* pattern specification of the *BooleanFormat* component.

BooleanFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *BooleanFormatter* class formats and parses **boolean** data.

BooleanFormatter constructors

BooleanFormatter()

public BooleanFormatter()

Constructs a *BooleanFormatter* object.

BooleanFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type, which is always *Variant.BOOLEAN* for *BooleanFormatter* classes.

BooleanFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class

Method	Implemented in
format(java.lang.Object)	com.borland.dx.text.Formatter
getSpecialObject(int)	com.borland.dx.text.Formatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[] , int, int)	com.borland.dx.text.Formatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.Formatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.Formatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.Formatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.Formatter
setPattern(java.lang.String)	com.borland.dx.text.Formatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.Formatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public final String format(Variant value)
```

Returns a *String* representing the given boolean value stored in the supplied *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

<i>value</i>	The value to be formatted to a <i>String</i> .
--------------	--

Overrides `com.borland.dx.text.VariantFormatter.format(Variant)`

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value. A **null** return value results when *stringValue* is **null** or empty.

<i>stringValue</i>	The string to be parsed.
--------------------	--------------------------

<i>value</i>	The <i>Variant</i> that receives the parsed result.
--------------	---

```
Overrides com.borland.dx.text.Formatter.parse(String, Variant)
```

ByteFormatter class

dx.text package

Extends com.borland.dx.text.IntegerFormatter

Implements java.io.Serializable

A formatter class for parsing and formatting byte data values. It is used for formatting and parsing **byte** columns in *DataSets*. The range of a **byte** is -128 to 127.

ByteFormatter variables

Variable	Defined in
type	com.borland.dx.text.IntegerFormatter

ByteFormatter constructors

ByteFormatter(int)

public ByteFormatter(int type)

Constructs a *ByteFormatter* object.

type The value of *type* must always be *Variant.BYTE*.

ByteFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	com.borland.dx.text.IntegerFormatter

ByteFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object

Method	Implemented in
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	com.borland.dx.text.IntegerFormatter
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

parse(com.borland.dx.dataset.Variant, char[], int, int)

public final void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.IntegerFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue

The string to be parsed.

value

The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.IntegerFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

DateFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *DateFormatter* class uses a *Variant* of type *Variant.DATE*, which stores data in a *java.sql.Date* object. This object is able to store time information; however, all time-related operations on *java.sql.Date* objects are deprecated.

To work with time data, use *TimeFormatter* or *TimestampFormatter*.

See also Edit/display mask patterns

DateFormatter constructors

DateFormatter()

```
public DateFormatter()
```

Constructs a *DateFormatter* object.

DateFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type, which is always *Variant.DATE* for *DateFormatter* classes.

DateFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public final String format(Variant value)
```

Returns a *String* representing the given date stored in the *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue The string to be parsed.

value The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String,
com.borland.dx.dataset.Variant)

DoubleFormatter class

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *DoubleFormatter* class formats and parses data of type **double**.

DoubleFormatter constructors

DoubleFormatter(int)

```
public DoubleFormatter(int type)
```

Constructs a *DoubleFormatter* object.

type The type of *Variant* data. *type* must always be *Variant.DOUBLE*.

DoubleFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type, which is always *Variant.DOUBLE* for a *DoubleFormatter*.

DoubleFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter

Method	Implemented in
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public final String format(Variant value)

Returns a *String* representing the double value stored in the *Variant*. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(Variant)

parse(java.lang.String, com.borland.dx.dataset.Variant)

public final void parse(String stringValue, Variant value)

Analyzes the given *String* and produces as output a *Variant* containing the appropriate double value.

stringValue The string to be parsed.

value The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(String, Variant)

IntegerFormatter class

dx.text package

Extends com.borland.dx.text.VariantFormatter

Extended by com.borland.dx.text.ByteFormatter, com.borland.dx.text.ShortFormatter

Implements java.io.Serializable

The *IntegerFormatter* class formats and parses data of type **int**.

type

For the *IntegerFormatter* class, *type* must always be *Variant.INTEGER*.

IntegerFormatter constructors

type

The value of *type* must always be *Variant.INTEGER*.

IntegerFormatter properties

variantType

Returns the *Variant* type.

IntegerFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public final String format(Variant value)

Returns a string representation of the **int** value stored in *Variant*. All reasonable attempts are made to “cast” the type found in the object to an **int**.

value The value that is formatted.

Overrides com.borland.dx.text.VariantFormatter.format(Variant)

parse(com.borland.dx.dataset.Variant, char[], int, int)

public void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.VariantFormatter.parse(Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

public void parse(String stringValue, Variant value)

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

<i>stringValue</i>	The string to be parsed.
<i>value</i>	The <i>Variant</i> that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(String, Variant)

InvalidFormatException class

dx.text package

Extends java.lang.Exception

Implements java.io.Serializable

The *InvalidFormatException* class is the *Exception* class generated by format and parser code exceptions. The *InvalidFormatException* is thrown for low level parsing conflicts, for example, entering “A” in a numeric field, and is generated when an edited data value is parsed. This is determined by the *editMask* and *displayMask* properties of a *Column* component.

The *InvalidFormatException* differs from the *ValidationException* in that the *InvalidFormatException* can only be triggered when editing is taking place. The *ValidationException* can be triggered whenever an invalid value is set and doesn’t depend on (keystroke) editing of data.

The *DataSet* class has special knowledge about *ValidationException* objects and automatically sends them to registered status listeners. Consequently, an *InvalidFormatException* caused by a parsing error is often turned into a *ValidationException*, and the *DataSet* sends it to the status listeners. As a

result, you typically only deal with the *ValidationException* class when editing a field.

See also [String-based patterns \(masks\)](#)

InvalidFormatException constructors

InvalidFormatException(java.lang.String)

```
public InvalidFormatException(String format)
```

Constructs an *InvalidFormatException* object that contains the editing error.

format The string describing the editing error that occurred.

InvalidFormatException(java.lang.String, int)

```
public InvalidFormatException(String format, int errorOffset)
```

Constructs an *InvalidFormatException* object that contains the editing error and the offset of the error.

format The string describing the editing error that occurred.

errorOffset The cursor position for this error.

InvalidFormatException properties

Property	Implemented in
class*	java.lang.Object
errorOffset*	this class
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable

errorOffset

```
public int getErrorOffset()
```

Returns the cursor position for this error.

fire(java.lang.String)

Throws an *InvalidFormatException*.

The string describing the error.

ItemEditMask interface

Implemented by com.borland.dx.text.ItemEditMaskStr

The *ItemEditMask* interface provides an open interface for character-by-character input validation. There is an *ItemEditMaskStr* implementation which uses a control string to validate characters.

ItemEditMask methods

Method	Implemented in
<code>delete(com.borland.dx.text.ItemEditMaskState, int, int)</code>	this class
<code>getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant, int)</code>	this class
<code>getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant)</code>	this class
<code>insert(com.borland.dx.text.ItemEditMaskState, char)</code>	this class
<code>isComplete(com.borland.dx.text.ItemEditMaskState)</code>	this class
<code>move(com.borland.dx.text.ItemEditMaskState, int)</code>	this class
<code>prepare(com.borland.dx.dataset.Variant)</code>	this class

delete(com.borland.dx.text.ItemEditMaskState, int, int)

`public boolean delete(ItemEditMaskState state, int startPos, int count)`

Deletes the given range of characters from the edit buffer and returns **true** if deletion occurred and the edit string has changed. If *delete()* returns false, the deletion could not take place.

Usually each deleted character is replaced with an underscore character.

<i>state</i>	The <i>ItemEditMaskState</i> returned by <i>prepare()</i> .
<i>startPos</i>	The starting position within the edit buffer (where 0 is the first character) to begin the deletion.
<i>count</i>	The number of characters to delete.

getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant)

`public void getFinalValue(ItemEditMaskState state, Variant value)`

Fetches the results from parsing the current edit buffer, storing the results into the *value* parameter. *getFinalValue()* never returns a **null** *Variant*, but it does throw an *InvalidFormatException* if the current edit buffer cannot be parsed. This exception class contains the cursor position where the failure occurred.

<i>state</i>	The state of the control returned by <i>prepare()</i> .
<i>value</i>	The <i>Variant</i> used to store the parsed results.

getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant, int)

public void getFinalValue(ItemEditMaskState state, Variant value, int variantType)

Fetches the results from parsing the current edit buffer, storing the results into the *value* parameter. *getFinalValue()* never returns a **null** *Variant*, but it does throw an *InvalidFormatException* if the current edit buffer cannot be parsed. This exception class contains the cursor position where the failure occurred.

state The state of the control returned by *prepare()*.
value The *Variant* used to store the parsed results.
variantType The data type for the *Variant*.

insert(com.borland.dx.text.ItemEditMaskState, char)

public boolean insert(ItemEditMaskState state, char c)

Inserts the specified character at the position given by *state.cursorPos*. If *insert()* returns **true**, the insert succeeded and the display string state is now different. If *insert()* returns **false**, the insert was refused. No error reporting or beeping occurs as the control is expected to do that.

state The *ItemEditMaskState* returned by *prepare()*.
c The character to be inserted. It is known that the character is not a navigation keystroke, but the implementor of this method must decide if the character is legal.

isComplete(com.borland.dx.text.ItemEditMaskState)

public boolean isComplete(ItemEditMaskState state)

Determines whether all the required fields in the edit buffer have been provided. If *isComplete* returns **true**, all required fields have been filled in. If it returns **false**, *state.cursorPos* is set at the first required character which has been left empty. This method does not perform validation.

state The *ItemEditMaskState* returned by *prepare()*.

move(com.borland.dx.text.ItemEditMaskState, int)

public boolean move(ItemEditMaskState state, int keyCode)

Handles the given navigation request starting from the specified cursor position. If the cursor position changes, *move()* returns **true**. If nothing happened, *move()* returns **false**.

state The *ItemEditMaskState* returned by *prepare()*.

keyCode Always one of the following:

- *Event.HOME*
- *Event.END*
- *Event.LEFT*
- *Event.RIGHT*
- *Event.MOUSE_DOWN*
- *Event.MOUSE_UP*

In the case of the two mouse events, the *state.cursorPos* contains the desired mouse position. This method should alter that position if it desires.

prepare(com.borland.dx.dataset.Variant)

public ItemEditMaskState prepare(Variant value)

The initial method called when setting up for editing against the *ItemEditMask* interface. It returns an *ItemEditMaskState*, an object allocated within the *prepare()* method but which should be owned by the control doing the editing. It contains state information regarding the current edit string and cursor position.

A **null** return value signifies that the *ItemEditMask* interface should not be used (meaning there will be no character-by-character checking during editing). *prepare()* instantiates some private data which is owned by the edit control.

value A *Variant* containing the data to be formatted into the edit buffer. A **null** value or *value.isNull()* specifies that the initial edit string is empty (except for embedded literals and underscore characters where characters should be entered). This *value* parameter is not recorded, so it can fall out of scope.

ItemEditMaskState component

dx.text package

Extends java.lang.Object

Implements java.io.Serializable

The *ItemEditMaskState* component carries state information for a control while it is using an *ItemEditMask* interface. The control owns this information, though it is instantiated by the *ItemEditMask*. This allows multiple controls to share a common *ItemEditMask*.

ItemEditMaskState variables

Variable	Defined in
cursorPos	this class
displayString	this class

cursorPos

public int cursorPos

The position of the cursor in the display string.

displayString

public StringBuffer displayString

The string the control displays to the user.

ItemEditMaskState constructors

ItemEditMaskState()

public ItemEditMaskState()

Constructs an *ItemEditMaskState* object. The default size of the string buffer is 16 characters, and the cursor is initially positioned at the beginning of the display string.

ItemEditMaskState(int, int)

public ItemEditMaskState(int size, int cursorPos)

Constructs an *ItemEditMaskState* object using the specified display string size and the specified cursor position in the display string.

size The size of the display string.

cursorPos The initial position of the cursor in the display string.

ItemEditMaskState properties

Property	Implemented in
class*	java.lang.Object

ItemEditMaskState methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

ItemEditMaskStr class

dx.text package

Extends java.lang.Object

Implements com.borland.dx.text.ItemEditMask,
com.borland.dx.text.ItemEditMaskRegionChar, java.io.Serializable

The *ItemEditMaskStr* class implements the *ItemEditMask* interface using pattern strings to control formatting, parsing, and edit interactions.

For information about constructing an edit mask, see string-based patterns.

ItemEditMaskStr constructors

ItemEditMaskStr(java.lang.String, com.borland.dx.text.VariantFormatter, int)

public ItemEditMaskStr(String editMask, VariantFormatter formatter, int variantType)

Constructs an *ItemEditMaskStr* object which implements a string-based *ItemEditMask*.

You do not need to construct an *ItemEditMask* for every text field, only those for which you want to constrain input on a character-by-character basis.

<i>editMask</i>	Contains a <i>String</i> which controls the character-by-character editing semantics when used by a text control. If null or empty, it inherits the <i>formatMask</i> from the <i>formatter</i> parameter.
<i>formatter</i>	The <i>VariantFormatter</i> object used by this class. If this parameter is null , a default one will be constructed from the other parameters.
<i>variantType</i>	Contains one of the values defined in <i>Variant</i> . <i>variantType</i> defines the type of data returned from the <i>getValue()</i> method. If it is zero, <i>variantType</i> defaults to that of the <i>formatter</i> .

See also *VariantFormatStr*, *ItemEditMask*

ItemEditMaskStr(java.lang.String, com.borland.dx.text.VariantFormatter, int, java.util.Locale)

public ItemEditMaskStr(String editMask, VariantFormatter formatter, int variantType, Locale locale)

Constructs an *ItemEditMaskStr* object which implements a string-based *ItemEditMask*.

You do not need to construct an *ItemEditMask* for every text field, only those for which you want to constrain input on a character-by-character basis.

<i>editMask</i>	Contains a <i>String</i> which controls the character-by-character editing semantics when used by a text control. If null or empty, it inherits the <i>formatMask</i> from the <i>formatter</i> parameter.
<i>formatter</i>	The <i>VariantFormatter</i> object used by this class. If this parameter is null , a default one will be constructed from the other parameters.

<i>variantType</i>	Contains one of the values defined in <i>Variant</i> . <i>variantType</i> defines the type of data returned from the <i>getValue()</i> method. If it is zero, <i>variantType</i> defaults to that of the <i>formatter</i> .
<i>locale</i>	Contains the locale to use. If null , the <i>Locale</i> of the <i>formatter</i> object is used. If <i>formatter</i> is also null , the current machine's default locale is used.

See also *VariantFormatStr*, *ItemEditMask*

ItemEditMaskStr properties

Property	Implemented in
class*	java.lang.Object

ItemEditMaskStr methods

Method	Implemented in
clone()	java.lang.Object
delete(com.borland.dx.text.ItemEditMaskState, int, int)	this class
deleteCharAt(java.lang.StringBuffer, int, char)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getCharAt(java.lang.StringBuffer, int)	this class
getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant, int)	this class
getFinalValue(com.borland.dx.text.ItemEditMaskState, com.borland.dx.dataset.Variant)	this class
hashCode()	java.lang.Object
insert(com.borland.dx.text.ItemEditMaskState, char)	this class
isComplete(com.borland.dx.text.ItemEditMaskState)	this class
isLiteral(int)	this class
isOptional(int)	this class
isPassword(int)	this class
isValid(int, char)	this class
literalAt(int)	this class
move(com.borland.dx.text.ItemEditMaskState, int)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
prepare(com.borland.dx.dataset.Variant)	this class
setCharAt(java.lang.StringBuffer, int, char)	this class

Method	Implemented in
shiftLeft(com.borland.dx.text.ItemEditMaskState)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

isPassword(int)

public boolean isPassword(int charPosition)

Returns **true** if the indicated character position in the edit mask is to be treated as a password character. This is determined by the presence of "*" (the password symbol) in the edit mask. See String-based patterns for details.

charPosition A character position in the edit mask.

literalAt(int)

public char literalAt(int charPosition)

Returns the literal character at the specified character position.

charPosition A character position in the edit mask.

shiftLeft(com.borland.dx.text.ItemEditMaskState)

protected boolean shiftLeft(ItemEditMaskState state)

Shifts the entire contents of the edit buffer left by one position. It stops when it hits an illegal situation (such as moving a letter into a digit field). It also stops at the first character in the string; that is, it doesn't drop characters off the left. It starts at the *state.cursorPos*. If that *cursorPos* is at a valid character, it converts that character into a *blankChar*. Blank characters are not shifted, so the first blank character stops the shift.

state The edit mask state of the control.

ItemFormatStr class

dx.text package

Extends com.borland.dx.text.ItemFormatter

ItemFormatStr extends the *ItemFormatter* class through the use of *String* patterns to control formatting and parsing. Though other implementations of *Formatter* are allowed, *ItemFormatStr* is the only one currently provided with *JDataStore*. Note that the current implementation supports only *Variants* and is only a wrapper for *VariantFormatStr*.

Four different kinds of pattern strings can be used. Each is distinct, and the fields from one cannot be used with another. The type used will be inferred from the *Variant.Type* passed into the constructor. The types are:

- *ItemFormatter.NUMERIC* (double)
- *ItemFormatter.DECIMAL* (BigDecimal)
- *ItemFormatter.DATETIME*
- *ItemFormatter.TEXT*

Numeric fields

The numeric format mask actually consists of two semicolon separated masks. The first is required. The second, if provided, will determine how negative numbers are formatted. For example, "###.##; (###.##) " will format negative numbers in parentheses.

The following table illustrates the characters allowed in numeric fields.

Table 6.2 Valid characters - numeric fields

Symbol	Meaning
0	A digit.
#	A digit, zero shows as absent.
.	Placeholder for decimal separator.
,	Placeholder for grouping delimiter. Shows the interval to be used.
;	Separates formats. There are two: positive and negative.
%	Divide by 100 and show as percentage.
X	Any other characters can be used in.
–	Leading minus for negative numbers.
()	Parenthesis to show the entire expression as negative (e.g. "(\$#,###.##)").
{}	Optional fields (e.g. "0000.{00}"). Will allow the decimal fraction to be omitted when editing.
^	Sets the initial cursor when editing.
\nnnn{}	Single character literal (e.g. \002, \0x0A, \u2030).

Date and timestamp fields

The following table illustrates the characters allowed in date or timestamp fields.

Table 6.3 Valid characters - date and timestamp fields

Symbol	Meaning	Presentation	Notes
G	Era designator	Text	AD/BC
y	Year	Number	Y,YY = 97, YYYY = 1997
M	Month in year	Text and number	M,MM = numeric, MMM = month abbrev, MMMM = full month

Table 6.3 Valid characters - date and timestamp fields (continued)

Symbol	Meaning	Presentation	Notes
d	Day in year	Number	d,dd = 10, dddd = 0010
h	Hour in am/pm (1~12)	Number	h,hh = 12, hhhh = 0012
H	Hour in day (0~23)	Number	H,HH = 23, HHHH = 0023
m	Minute in hour (0~23)	Number	m,mm = 59, mmmm = 0059
s	Second in minute	Number	s,ss = 59, ssss = 0059
S	Millisecond	Number	S = 9, SS = 99, SSS = 999
E	Day in week	Text	E,EE,EEE = Sun, EEEE = Sunday
D	Day in year	Number	D,DD = 364, DDDD = 0364
F	Day of week in month	Number	2 (2nd Wed in July)
w	Week in year	Number	27
W	Week in month	Number	2
a	AM/PM marker	Text	AM/PM
k	Hour in day (1~24)	Number	24
K	Hour in am/pm (0~11)	Number	0
z	Time zone	Text	z,zz,zzz = PDT, zzzz = Pacific Standard Time
'	Escape for text		
"	Single quote		
\nnnn	Single character literal (e.g. \002, \0x0A, \u2030)		
()	Can be used to bracket optional fields (e.g. "0000.{00}" will allow the decimal fraction to be omitted when editing)		
^	Sets the initial cursor position when editing		

String fields

The string editmask can actually consist of up to four distinct subfields, separated by the semicolons. The subfields are:

- The edit mask itself
- A '0' or '1' indicating whether literals should be stripped ('0' = strip out literals)
- The character to use as a "blank" indicator. This is used to show the user which elements have not yet been entered.
- The character to be used to replace blank characters on output. If there is no character given, blank characters are stripped.

For example, the edit mask "(999)000-0000;0;_" would indicate:

- The template the user would see on field entry would be "(____)____-____"
- Unfilled characters will be shown by "_" (underscore)
- Literals will be removed on output, so (408)555-1234 would become 4085551234
- Blank characters will be stripped, so (____)555-1234 would become 5551234

Not all three subfields are required. If subfield 3 is omitted, the "_" (underscore) character will be the blank indicator. If subfield 2 is omitted, literals are not removed.

The following table illustrates the characters allowed in string fields.

Table 6.4 Valid characters - string fields

Symbol	Meaning
0	Digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).
9	Digit or space (entry not required; plus and minus signs not allowed).
#	Digit or space (entry not required; blank positions converted to spaces, plus and minus signs allowed).
L	Letter (A through Z, entry required).
l	Letter (A through Z, entry optional).
?	Letter (A through Z, entry optional).
A	Letter or digit (entry required).
a	Letter or digit (entry optional).
C	Any character or a space (entry optional).
c	Any character or a space (entry required).
&	Any character or a space (entry required).
<	Causes all characters that follow to be converted to lowercase.
>	Causes all characters that follow to be converted to uppercase.
!	Causes input mask to fill from right to left, rather than from left to right when characters on the left side of the input mask are optional. The exclamation point can be included anywhere in the input mask.
\	Causes the character that follows to be displayed as a literal character. Used to display any of the characters listed in this table as literal characters (for example, \A is displayed as just A).
\nnnn	Single character literal (e.g. \002, \0x0A, \u2030).
"	Encloses a literal expression (for example, the pattern "990' units sold'" would display as "27 units sold").
**	Encloses a password encrypted string. For example, the pattern "**AAAAaaaa*" would accept a password of at least 4, and at most 8, alphanumeric characters. The characters would echo as "*" as they were typed.
{}	Can be used to bracket optional fields. For example, "LLLL{LLL}" will allow a user to edit at least (but optionally up to 7) letters.
^	Sets the initial cursor position when editing.

ItemFormatStr constructors

ItemFormatStr(java.lang.String, int)

public ItemFormatStr(String pattern, int variantType)

Constructs a string-based implementation of the Formatter interface.

<i>pattern</i>	The string of special characters used to format values when using the <i>format()</i> method. If this value is null (or empty), the best "default" pattern will be selected based on the locale.
<i>variantType</i>	<div>The type of pattern being used. This can be any one of the following 4 types:<ul style="list-style-type: none">• <i>ItemFormatter.NUMERIC</i>• <i>ItemFormatter.DATETIME</i>• <i>ItemFormatter.TEXT</i>• <i>ItemFormatter.DECIMAL</i>A value of zero will default to <i>ItemFormatter.TEXT</i>.</div>

ItemFormatStr(java.lang.String, int, java.util.Locale)

public ItemFormatStr(String pattern, int variantType, Locale locale)

Constructs a string-based implementation of the Formatter interface.

<i>pattern</i>	The string of special characters used to format values when using the <i>format()</i> method. If this value is null (or empty), the best "default" pattern will be selected based on the locale.
<i>variantType</i>	<div>The type of pattern being used. This can be any one of the following 4 types:<ul style="list-style-type: none">• <i>ItemFormatter.NUMERIC</i>• <i>ItemFormatter.DATETIME</i>• <i>ItemFormatter.TEXT</i>• <i>ItemFormatter.DECIMAL</i>A value of zero will default to <i>ItemFormatter.TEXT</i>.</div>
<i>locale</i>	The locale to control this pattern. This will determine things like the decimal point sign, the currency sign, etc. If this value is null , the current default locale will be used.

ItemFormatStr properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class

Property	Implemented in
locale*	this class
pattern*	this class

formatObj

public Format getFormatObj()

Returns the JDK Format subclass associated with this *ItemFormatter*.

getFormatObj() returns **null** if the constructor could not accept the initial pattern.

locale

public Locale getLocale()

Returns the *Locale* currently being used by this Formatter. Currently, there is no way to change this locale once the Formatter has been created. The returned value is never **null**.

pattern

public String getPattern()

Returns the pattern currently being used by this Formatter for parsing and formatting.

ItemFormatStr methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(java.lang.Object)	this class
getSpecialObject(int)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(java.lang.String)	this class
setPattern(java.lang.String)	this class
setSpecialObject(int, java.lang.Object)	this class
toString()	java.lang.Object
wait()	java.lang.Object

Method	Implemented in
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(java.lang.Object)

public String format(Object value)

Returns a *String* representing the given value stored in the supplied *Variant*. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.ItemFormatter.format(java.lang.Object)

getSpecialObject(int)

public Object getSpecialObject(int objType)

Returns the value of the specified special object.

Some Formatter classes define special objects for their own use. You must know the internal details of the Format subclass being used to use *getSpecialObject()*.

objType Contains an identifier telling which object in which Formatter to retrieve. May be either *ItemFormatter.FILLCHARACTER* or *ItemFormatter.REPLACECHARACTER*. Fill characters are used to fill empty slots in the string. Replace characters replace fill characters on output.

Overrides com.borland.dx.text.ItemFormatter.getSpecialObject(int)

parse(java.lang.String)

public Object parse(String stringValue)

Analyzes the given *String* and produces as output an *Object* containing the appropriate value. A **null** return value results when *stringValue* is **null** or empty.

stringValue The string to be parsed.

Overrides com.borland.dx.text.ItemFormatter.parse(java.lang.String)

setPattern(java.lang.String)

```
public String setPattern(String pattern)
```

Sets the pattern used for parsing and formatting to a new pattern, returning the old pattern. The new pattern must be of the same basic type associated with this type of Formatter. For example, if you used a Date/Time pattern in the constructor, you can't switch to a numeric pattern as each basic pattern type has its own data-dependent *format()* and *parse()* methods.

If the new pattern is **null** (or empty), *setPattern()* chooses a default pattern for the current locale.

pattern The new pattern to be used for formatting and parsing.

Overrides com.borland.dx.text.ItemFormatter.setPattern(java.lang.String)

setSpecialObject(int, java.lang.Object)

```
public Object setSpecialObject(int objType, Object obj)
```

Sets the special object associated with a particular Formatter. This is a general purpose routine to set specific booleans, characters, flags, and so on inside a formatter, but it is completely dependent on the formatter being used. *setSpecialObject()* returns the prior special object, which can be useful for restoring the original value after a temporary alteration.

objType Identifies which object in which formatter to retrieve. It can be *ItemFormatter.FILLCHARACTER* or *ItemFormatter.REPLACECHARACTER*. Fill characters are used to fill empty slots in the string. Replace characters replace fill characters on output.

obj Contains the object to be set. The type of the *Object* must match the expected type for the given *objType*; do not pass a **null** object.

ItemFormatter class (abstract)

dx.text package

Extends java.lang.Object

Extended by com.borland.dx.text.VariantFormatter, com.borland.dx.text.ItemFormatStr
ItemFormatter is an abstract class that is the superclass of several others used to format and parse various types of data. Because different data types typically require different kinds of handling, there are five basic parse/format types currently supported:

- Numeric
- Currency
- Date and time
- Text
- Boolean

ItemFormatter has defined constants for each of these:

- NUMERIC = 1
- DECIMAL = 2
- DATETIME= 3
- TEXT = 4
- BOOLEAN = 5

All Formatter classes have a *format()* method that returns a data object as a string, and they have a *parse()* method that analyzes a string value and returns a data object.

The *ItemFormatter* class has a *pattern* property used to access the edit/display mask patterns that are used to format and parse the data.

The most commonly used subclass is *ItemFormatStr*, which uses pattern strings based on the JDK Format conventions.

ItemFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class
locale*	this class
pattern*	this class

formatObj

public Format getFormatObj()

ItemFormatter is itself a layer on the JDK's Format interface. *getFormatObj()* provides access to the underlying Format object being used by a particular Formatter (which depends on the type of data being formatted). It returns the Format object being used (see JDK's description of Format, NumberFormat, DecimalFormat, and SimpleTimeFormat).

getFormatObj() returns **null** if the constructor could not accept the initial pattern.

locale

public Locale getLocale()

Returns the *Locale* currently being used by this Formatter. Currently, there is no way to change this locale once the Formatter has been created. The returned value is never **null**.

pattern

public String getPattern()

Returns the pattern currently being used by this Formatter for parsing and formatting.

ItemFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(java.lang.Object)	this class
getSpecialObject(int)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(java.lang.String)	this class
setPattern(java.lang.String)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(java.lang.Object)

public abstract String format(Object value)

Returns a *String* representing the given value stored in the supplied object. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

getSpecialObject(int)

public Object getSpecialObject(int objType)

Returns the value of the specified special object.

LongFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the Variant type, which is always *Variant.LONG* for *LongFormatter*.

LongFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter

Method	Implemented in
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public final String format(Variant value)

Returns a *String* representing the given long value stored in the *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(com.borland.dx.dataset.Variant, char[], int, int)

public final void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

variant The parsed value (cannot be **null**).
value The character array containing the text to parse.
offset The zero-based offset into the character array.
len The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.VariantFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

public final void parse(String stringValue, Variant value)

Analyzes the given *String* and produces as output an *Object* containing the appropriate value. A **null** return value results when *stringValue* is **null** or empty.

stringValue The string to be parsed.
value The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

ObjectFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *ObjectFormatter* component formats and parses data of type *Object*.

ObjectFormatter constructors

ObjectFormatter()

public ObjectFormatter()

Creates an *ObjectFormatter* class.

ObjectFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the Variant type, which is always *Variant.OBJECT* for *ObjectFormatter*.

ObjectFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object

Method	Implemented in
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public final String format(Variant value)

Returns a *String* representing the given value stored in the *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(com.borland.dx.dataset.Variant, char[], int, int)

```
public void parse(Variant variant, char[] value, int offset, int len)
```

A high-speed parse that parses directly into a character array.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

Overrides `com.borland.dx.text.VariantFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)`

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value. A **null** return value results when *stringValue* is **null** or empty.

<i>stringValue</i>	The string to be parsed.
<i>value</i>	The <i>Variant</i> that receives the parsed result.

Overrides `com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)`

ShortFormatter class

dx.text package

Extends `com.borland.dx.text.IntegerFormatter`

Implements `java.io.Serializable`

A Formatter class for parsing and formatting **short** data types. *ShortFormatter* is used for short columns, handling values within the range of 32767 to -32768.

ShortFormatter variables

Variable	Defined in
type	<code>com.borland.dx.text.IntegerFormatter</code>

ShortFormatter constructors

ShortFormatter(int)

public ShortFormatter(int type)

Constructs a *ShortFormatter* object.

type The value of *type* must always be *Variant.SHORT* for *ShortFormatter*.

ShortFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	com.borland.dx.text.IntegerFormatter

ShortFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	com.borland.dx.text.IntegerFormatter
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter

Method	Implemented in
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

parse(com.borland.dx.dataset.Variant, char[], int, int)

```
public final void parse(Variant variant, char[] value, int offset, int len)
```

Analyzes the text in a character array and produces as output a *Variant* containing the parsed value.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.IntegerFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

<i>stringValue</i>	The string to be parsed.
<i>value</i>	The <i>Variant</i> that receives the parsed result.

Overrides com.borland.dx.text.IntegerFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

SimpleFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *SimpleFormatter* component is a wrapper for the *VariantFormatStr* class. It is a simple implementation of a formatting and parsing class that uses the default locale and the default control pattern for the particular *Variant* that is passed to it.

See also Edit/display mask patterns

SimpleFormatter constructors

SimpleFormatter()

public SimpleFormatter()

Constructs a *SimpleFormatter* object.

SimpleFormatter(int)

public SimpleFormatter(int variantType)

Constructs a *SimpleFormatter* object that instantiates a *VariantFormatStr* object of the specified *Variant* type.

variantType The *Variant* type of the data the class will format and parse.

SimpleFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class
locale*	this class
pattern*	this class
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

formatObj

public Format getFormatObj()

Calls *getFormatObj()* of the *VariantFormatStr* object, returning the format object.

locale

public Locale getLocale()

Calls *getLocale()* of the *VariantFormatStr* object, returning the locale.

pattern

public String getPattern()

Calls *getPattern()* of the *VariantFormatStr* object, returning the default pattern.

variantType

public int getVariantType()

Calls *getVariantType()* of the *VariantFormatStr* object, returning the *Variant* type.

SimpleFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter

Method	Implemented in
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	this class
setSpecialObject(int, java.lang.Object)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public String format(Variant value)
```

Calls the *format()* method of the *VariantFormatStr* object, returning the formatted string.

<i>value</i>	The value to be formatted. If <i>value</i> isn't a <i>Variant</i> , the method throws the <i>InvalidFormatException</i> .
--------------	---

Overrides `com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)`

getSpecialObject(int)

```
public Object getSpecialObject(int objType)
```

Calls the *getSpecialObject()* method of the *VariantFormatStr* object, returning the special object.

Some Formatter classes define special objects for their own use. You must know the internal details of the Format subclass being used to use *getSpecialObject()*.

<i>objType</i>	The special object type to return.
----------------	------------------------------------

Overrides `com.borland.dx.text.VariantFormatter.getSpecialObject(int)`

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

<i>stringValue</i>	The string to be parsed.
--------------------	--------------------------

<i>value</i>	The <i>Variant</i> that receives the parsed result.
--------------	---

```
Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String,  
com.borland.dx.dataset.Variant)
```

parse(java.lang.String, com.borland.dx.dataset.Variant, int)

public void parse(String stringValue, Variant value, int variantType)

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

<i>stringValue</i>	The string to be parsed.
<i>value</i>	The <i>Variant</i> that receives the parsed result.
<i>variantType</i>	The type of <i>Variant</i> being used.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant, int)

setPattern(java.lang.String)

public String setPattern(String pattern)

Sets the pattern used for parsing and formatting to a new pattern, returning the old pattern. The new pattern must be of the same basic type associated with this type of formatter. For example, if you used a Date/Time pattern in the constructor, you can't switch to a numeric pattern as each basic pattern type has its own data-dependent *format()* and *parse()* methods.

If the new pattern is **null** (or empty), *setPattern()* chooses a default pattern for the current locale.

<i>pattern</i>	The new pattern to be used for formatting and parsing.
----------------	--

Overrides com.borland.dx.text.VariantFormatter.setPattern(java.lang.String)

setSpecialObject(int, java.lang.Object)

public Object setSpecialObject(int objType, Object obj)

Some Formatter classes define special objects for their own use. This method allows them to be set. You must know the internal details of the Format subclass being used to use *setSpecialObject()*.

The returned value is the prior value of the object.

<i>charType</i>	The special object type. The possible values are <i>VariantFormatter.FillChar</i> , which is the fill character to fill blank slots, and <i>VariantFormatter.Replacecharacter</i> , which is used to replace <i>FillChar</i> on parse.
-----------------	--

<i>obj</i>	The special object.
------------	---------------------

Overrides com.borland.dx.text.VariantFormatter.setSpecialObject(int, java.lang.Object)

StringFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *StringFormatter* component formats and parses string data. It uses the *pattern* property of *ItemFormatter* to access the edit/display mask patterns that are used to format and parse the data.

See also Edit/display mask patterns

StringFormatter constructors

StringFormatter()

public StringFormatter()

Constructs a *StringFormatter* object.

StringFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type of *StringFormatter*, which is always *Variant.STRING* for *StringFormatter*.

StringFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object

Method	Implemented in
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public final String format(Variant value)
```

Returns a *String* representing the given value stored in the supplied object. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

<i>value</i>	The value to be formatted to a <i>String</i> .
--------------	--

```
Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)
```

parse(com.borland.dx.dataset.Variant, char[], int, int)

public void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.VariantFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

public void parse(String stringValue, Variant value)

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

<i>stringValue</i>	The string to be parsed.
<i>value</i>	The <i>Variant</i> that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

TextFormat component

dx.text package

Extends java.text.Format

Implements java.io.Serializable, java.lang.Cloneable

This component extends the basic *Format* class but allows for special formatting of text. Similar to other *Format* derivatives, it is based on a control pattern. The format of the control pattern is as follows:

<pattern>; <keepLiterals>; <fillCharacter>; <replaceCharacter>

pattern

Can contain any of the following characters (they are a common format shared by many products):

- 0 Digit 0:9, entry required, '+' and '-' not allowed
- 9 Digit 0:9, entry optional, '+' and '-' not allowed
- # Digit or space, entry optional, plus and minus signs allowed
- L Letter A:Z, entry required
- l Letter A:Z, entry optional
- ? Letter A:Z, entry optional
- A Letter A:Z or digit 0:9, entry required
- a letter A:Z or digit 0:9, entry optional
- C any character or space, entry required
- c Any character or space, entry optional
- & Any character or space, entry required
- < Causes all characters following to be converted to lowercase
- > Causes all characters following to be converted to uppercase
- ! Causes strings too short to fill from right to left
- \ Backslash escape -- allows any Unicode value to follow (e.g. "\u2003")
- ^ Initial place for cursor when editing begins
- ' Encloses a literal expression (for example, the pattern "990' units sold'" would display as "27 units sold")
- * Encloses a password encrypted string (for example the pattern "*AAAAaaaa*" would accept a password at least 4, and at most 8 alphanumeric characters). Characters typed or displayed appear as the '*' character

An example of a US phone number might be “!(999)000-0000”

keepLiterals

If this value is “0”, then literals in the string are removed before the value is stored. For example (408)555-5330 becomes 4085555330. Any other value (including no value at all) defaults to “1”, which means that literals are preserved.

fillCharacter

Whenever a string is formatted that is too short to fill all the available positions in the pattern, this character is used to fill the extra space(s). For example, formatting “()555-5330” with the pattern “!(999)000-0000;1;*” produces “(***)555-5330”. Note that a value of zero (indicated by “\0”) means that no filling occurs.

replaceCharacter

Whenever a string is parsed, all occurrences of *fillCharacter* are replaced with *replaceCharacter*. Using the example above, parsing “(***)555-5330” with the pattern “!(999)000-0000;1;*_” produces “(____)555-5330”. Again, a zero value is allowed in this field (using “\0”) and has the effect of removing the *fillCharacters*.

Note Each of the special fields (*keepLiterals*, *fillCharacter*, *replaceCharacter*) have their own setter and getter methods.

TextFormat variables

Variable	Defined in
NOT_A_CHAR	this class

NOT_A_CHAR

public static char NOT_A_CHAR = 0xffff

Not a valid character.

TextFormat constructors

TextFormat()

public TextFormat()

Constructs a *TextFormat* object with no specified string pattern.

TextFormat(java.lang.String)

public TextFormat(String pattern)

Constructs a *TextFormat* object with the specified string pattern.

pattern The pattern to construct the *TextFormat* object with.

TextFormat properties

Property	Implemented in
class*	java.lang.Object
fillCharacter	this class
keepLiterals	this class
replaceCharacter	this class

fillCharacter

public char getFillCharacter()

public void setFillCharacter(char c)

Retrieves and sets the fill character used in the string.

c The character that is used to fill spaces when the string is too short to fill all available spaces.

keepLiterals

```
public boolean getKeepLiterals()
public void setKeepLiterals(boolean tf)
```

Determines whether literals remain in the string when it is stored. *keepLiterals* is **true** when literals are stored; otherwise, it is **false**.

tf Set *tf* to **true** to store literals; set *tf* to **false** remove literals when the string is stored.

replaceCharacter

```
public char getReplaceCharacter()
public void setReplaceCharacter(char c)
```

Retrieves and sets the character that replaces the fill character when the string is parsed.

c The character that replaces the fill character when the string is parsed. A “\0” value results in removing all fill characters in the string.

TextFormat methods

Method	Implemented in
applyPattern(java.lang.String)	this class
clone()	java.text.Format
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(java.lang.Object, java.lang.StringBuffer, java.text.FieldPosition)	this class
format(java.lang.Object)	java.text.Format
format(java.lang.String, java.lang.StringBuffer, java.text.FieldPosition)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(java.lang.String, java.text.ParsePosition)	this class
parseObject(java.lang.String, java.text.ParsePosition)	this class
parseObject(java.lang.String)	java.text.Format
toPattern()	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

applyPattern(java.lang.String)

```
public void applyPattern(String pattern)
```

Sets the pattern to the specified value.

pattern The value to set the pattern to.

format(java.lang.Object, java.lang.StringBuffer, java.text.FieldPosition)

```
public final StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos)
```

Formats the given *Object* using the pattern associated with this object.

obj The *Object* to be formatted.

toAppendTo The *StringBuffer* to append the newly formatted *String* to.

pos The starting position in the *StringBuffer* of the formatted *String*.

Overrides `java.text.Format.format(java.lang.Object, java.lang.StringBuffer, java.text.FieldPosition)`

format(java.lang.String, java.lang.StringBuffer, java.text.FieldPosition)

```
public StringBuffer format(String toBeFormatted, StringBuffer result, FieldPosition pos)
```

Formats the given *String* using the pattern associated with this object. If the input string has insufficient characters to fill the pattern, it is filled with the character indicated by *setFillChar()*. The *result* parameter is assigned the return value of this method.

toBeFormatted The *String* to be formatted.

result The formatted string.

pos The starting position in the *StringBuffer* of the formatted *String*.

parse(java.lang.String, java.text.ParsePosition)

```
public StringBuffer parse(String text, ParsePosition pos)
```

This method parses (or decomposes) a *String* using the existing pattern. It allocates a new *StringBuffer* and fills it with the parsed version of the *text* parameter.

text The *String* to parse.

pos The starting position in the *StringBuffer* of the parsed text.

parseObject(java.lang.String, java.text.ParsePosition)

public final Object parseObject(String source, ParsePosition pos)

Parses (or decomposes) a *String* into an *Object*.

source

The *String* to parse.

pos

The starting position in the *StringBuffer* of the formatted *String*.

toPattern()

public String toPattern()

Returns the pattern used for formatting.

TimeFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *TimeFormatter* class formats and parses time data values.

See also Edit/display mask patterns

TimeFormatter constructors

TimeFormatter()

public TimeFormatter()

Constructs a *TimeFormatter* object.

TimeFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the type of the *Variant*, which is always *Variant.TIME* for *TimeFormatter*.

TimeFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public final String format(Variant value)
```

Returns a *String* representing the given timestamp stored in the *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(com.borland.dx.dataset.Variant, char[], int, int)

```
public void parse(Variant variant, char[] value, int offset, int len)
```

A high-speed parse that parses directly into a character array.

variant The parsed value (cannot be **null**).

value The character array containing the text to parse.

offset The zero-based offset into the character array.

len The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.VariantFormatter.parse(com.borland.dx.dataset.Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue The string to be parsed.

value The *Variant* that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

TimestampFormatter component

dx.text package

Extends com.borland.dx.text.VariantFormatter

Implements java.io.Serializable

The *TimestampFormatter* class formats and parses timestamp data.

See also Edit/display mask patterns

TimestampFormatter constructors

TimestampFormatter()

public TimestampFormatter()

Constructs a *TimestampFormatter* object.

TimestampFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	com.borland.dx.text.VariantFormatter
locale*	com.borland.dx.text.VariantFormatter
pattern*	com.borland.dx.text.VariantFormatter
scale*	com.borland.dx.text.VariantFormatter
variantType*	this class

variantType

public int getVariantType()

Returns the *Variant* type, which is always *Variant.TIME* for *TimestampFormatter*.

TimestampFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getSpecialObject(int)	com.borland.dx.text.VariantFormatter
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	com.borland.dx.text.VariantFormatter

Method	Implemented in
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	com.borland.dx.text.VariantFormatter
setSpecialObject(int, java.lang.Object)	com.borland.dx.text.VariantFormatter
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

public final String format(Variant value)

Returns a *String* representing the given timestamp stored in the *Variant*. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

value The value to be formatted to a *String*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

parse(com.borland.dx.dataset.Variant, char[], int, int)

public void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

variant The parsed value (cannot be **null**).

value The character array containing the text to parse.

offset The zero-based offset into the character array.

len The maximum number of characters in the array to use in the parse.

Overrides com.borland.dx.text.VariantFormatter.parse(Variant, char[], int, int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public final void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue The string to be parsed.

value The Variant that receives the parsed result.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String,
com.borland.dx.dataset.Variant)

VariantFormatStr class

dx.text package

Extends com.borland.dx.text.VariantFormatter

The *VariantFormatStr* class extends the *VariantFormatter* class through the use of *String* patterns to control formatting and parsing. It can handle all *Variant* types. Other formatter classes exist that are built to handle a specific *Variant* type.

There are five different kinds of pattern strings which can be used. Each is distinct, and the fields from one cannot be used with another. The type used will be inferred from the *Variant*.type passed into the constructor. The types are:

- ItemFormatter.NUMERIC (double)
- ItemFormatter.DECIMAL (BigDecimal)
- ItemFormatter.DATETIME
- ItemFormatter.TEXT
- ItemFormatter.BOOLEAN

See edit/display masks for information about using the various patterns.

VariantFormatStr constructors

VariantFormatStr(java.lang.String, int)

```
public VariantFormatStr(String pattern, int variantType)
```

Constructs a *VariantFormatStr* object that specifies the pattern used to format values and the type of data the object formats and parses.

pattern The string of special characters uses to format values when using the *format()* method. If this value is **null** (or empty), the best default pattern is automatically selected based on the locale.

variantType The *Variant* data type that is used by the *format()* and *parse()* methods.

VariantFormatStr(java.lang.String, int, java.util.Locale)

public VariantFormatStr(String pattern, int variantType, Locale locale)

Constructs a *VariantFormatStr* object that specifies the pattern used to format values, the type of data the object formats and parses, and the locale.

<i>pattern</i>	The string of special characters uses to format values when using the <i>format()</i> method. If this value is null (or empty), the best default pattern is automatically selected based on the locale.
<i>variantType</i>	The <i>Variant</i> data type that is used by the <i>format()</i> and <i>parse()</i> methods.
<i>locale</i>	The locale, which affects such things as what the currency sign looks like, how time and date data appears, and so on. If <i>locale</i> is null , the current default locale is used.

VariantFormatStr(java.lang.String, int, java.util.Locale, int, int, boolean)

public VariantFormatStr(String pattern, int variantType, Locale locale, int scale, int precision, boolean isCurrency)

Constructs a *VariantFormatStr* object that specifies the pattern used to format values, the type of data the object formats and parses, the locale, and other formatting information.

<i>pattern</i>	The string of special characters uses to format values when using the <i>format()</i> method. If this value is null (or empty), the best default pattern is automatically selected based on the locale.
<i>variantType</i>	The <i>Variant</i> data type that is used by the <i>format()</i> and <i>parse()</i> methods.
<i>locale</i>	The locale, which affects such things as what the currency sign looks like, how time and date data appears, and so on. If <i>locale</i> is null , the current default locale is used.
<i>scale</i>	Used for <i>BigDecimal</i> data types. Any value other than -1 selects the number of decimal digits used in formatting and parsing <i>BigDecimal</i> values. Note that <i>pattern</i> must still express the number of digits to be displayed.
<i>precision</i>	Must be -1 . Currently, <i>precision</i> is not used.
<i>isCurrency</i>	Indicates that a numeric field is currency. If <i>isCurrency</i> is true , the data is a currency value.

VariantFormatStr properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class
locale*	this class
pattern*	this class
scale*	this class
variantType*	this class

formatObj

public Format getFormatObj()

Returns the JDK *Format* subclass associated with this formatter. A returned value of **null** is possible.

locale

public Locale getLocale()

Returns the *Locale* object being used by this formatting class. The returned value will never be **null**.

pattern

public String getPattern()

Returns the pattern used by this *VariantFormatStr* object for formatting and parsing. See edit/display patterns.

scale

public int getScale()

Returns the current scale factor, which is relevant only for *BigDecimal* data.

variantType

public int getVariantType()

Returns the *Variant* type for this *VariantFormatStr* object.

VariantFormatStr methods

Method	Implemented in
buildTrueFormatMask(java.lang.String)	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	com.borland.dx.text.VariantFormatter
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	com.borland.dx.text.VariantFormatter
getDefaultPattern(int)	this class
getSpecialObject(int)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	com.borland.dx.text.VariantFormatter
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	com.borland.dx.text.VariantFormatter
setFromDouble (com.borland.dx.dataset.Variant, int, double)	com.borland.dx.text.VariantFormatter
setFromInt(com.borland.dx.dataset.Variant, int, int)	com.borland.dx.text.VariantFormatter
setPattern(java.lang.String)	this class
setSpecialObject(int, java.lang.Object)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

buildTrueFormatMask(java.lang.String)

public static final String buildTrueFormatMask(String editMask)

Returns a pattern string, removing any borland.com-specific extensions.

edit

The edit mask to be used for formatting and parsing the data.

format(com.borland.dx.dataset.Variant)

public String format(Variant value)

Returns a formatted string from the value specified with the *value* parameter. *format()* uses the current pattern to format the value. *format()* does not support all data types, but an attempt is made to cast the *Variant* data into a type required by the formatting logic.

The returned formatted string could be empty if the input *Variant* was **null** or unassigned. A **null** is returned if the formatting fails.

value The value to be formatted. If *value* isn't a *Variant*, the method throws the *InvalidFormatException*.

Overrides com.borland.dx.text.VariantFormatter.format(com.borland.dx.dataset.Variant)

getDefaultPattern(int)

protected String getDefaultPattern(int variantType)

Returns the default pattern used for formatting the data based on the *Variant* type and the current locale.

variantType The *Variant* data type.

getSpecialObject(int)

public Object getSpecialObject(int objType)

Retrieves the special object associated with a particular formatter. This is a general purpose routine to obtain specific booleans, characters, flags, and so on inside a formatter, but it is completely dependent on the formatter being used.

objType Identifies which object in which formatter to retrieve. It can be *ItemFormatter.FILLCHARACTER* or *ItemFormatter.REPLACECHARACTER*. Fill characters are used to fill empty slots in the string. Replace characters replace fill characters on output.

Overrides com.borland.dx.text.VariantFormatter.getSpecialObject(int)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public void parse(String stringValue, Variant value)
```

Parses a string using the current pattern and produces the appropriate value in the form of a *Variant*.

stringValue The string to be parsed. A null or empty value returns a *Variant* object which is set to *AssignedNull*.

value The *Variant* that contains the result. Its type is determined by the *variantType* parameter in the class constructor.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant)

parse(java.lang.String, com.borland.dx.dataset.Variant, int)

```
public void parse(String stringValue, Variant value, int variantType)
```

Parses a string using the current pattern and produces the appropriate value in the form of a *Variant*.

stringValue The string to be parsed. A null or empty value returns a *Variant* object which is set to *AssignedNull*.

value The *Variant* that contains the result.

variantType Specifies the *Variant* type returned in the value parameter.

Overrides com.borland.dx.text.VariantFormatter.parse(java.lang.String, com.borland.dx.dataset.Variant, int)

setPattern(java.lang.String)

```
public String setPattern(String pattern)
```

Sets the pattern used by this *VariantFormatStr* object to format and parse. See edit/display masks for information about specifying a pattern.

The prior pattern is returned.

pattern The pattern that controls the formatting and parsing.

Overrides com.borland.dx.text.VariantFormatter.setPattern(java.lang.String)

setSpecialObject(int, java.lang.Object)

```
public Object setSpecialObject(int objType, Object obj)
```

Sets the special object associated with a particular formatter. This is a general purpose routine to obtain specific booleans, characters, flags, and so on inside a formatter, but it is completely dependent on the formatter being

used. *setSpecialObject()* returns the prior special object, which can be useful for restoring the original value after a temporary alteration.

- objType*

Identifies which object in which formatter to retrieve. It can be *ItemFormatter.FILLCHARACTER* or *ItemFormatter.REPLACECHARACTER*. Fill characters are used to fill empty slots in the string. Replace characters replace fill characters on output.
- obj*

Contains the object to be set. The type of the *Object* must match the expected type for the given *objType*; do not pass a **null** object.

Overrides com.borland.dx.text.VariantFormatter.setSpecialObject(int, java.lang.Object)

VariantFormatter class (abstract)

dx.text package

Extends com.borland.dx.text.ItemFormatter

Extended by com.borland.dx.text.BigDecimalFormatter,
com.borland.dx.text.BinaryFormatter,
com.borland.dx.text.BooleanFormatter,
com.borland.dx.text.IntegerFormatter, com.borland.dx.text.DateFormatter,
com.borland.dx.text.DoubleFormatter, com.borland.dx.text.LongFormatter,
com.borland.dx.text.ObjectFormatter, com.borland.dx.text.SimpleFormatter,
com.borland.dx.text.StringFormatter, com.borland.dx.text.TimeFormatter,
com.borland.dx.text.TimestampFormatter,
com.borland.dx.text.VariantFormatStr

This general-purpose formatting class is a subclass of the abstract *ItemFormatter* that formats and parses *Variant* data. All classes in the *dx.dataset* package use the *VariantFormatter* class exclusively.

VariantFormatter properties

Property	Implemented in
class*	java.lang.Object
formatObj*	this class
locale*	this class
pattern*	this class
scale*	this class
variantType*	this class

formatObj

public Format getFormatObj()

VariantFormatter is itself a layer on the JDK's *Format* interface. *getFormatObj()* provides access to the underlying *Format* object being used by a particular *Formatter* (which depends on the type of data being formatted). It returns the *Format* object being used (see JDK's description of *Format*, *NumberFormat*, *DecimalFormat*, and *SimpleTimeFormat*). *getFormatObj()* returns **null** if the constructor could not accept the initial pattern.

locale

public Locale getLocale()

Returns the *Locale* currently being used by this *Formatter*. Currently, there is no way to change this locale once the *Formatter* has been created. The returned value is never **null**.

pattern

public String getPattern()

Returns the pattern currently being used by this *Formatter* for parsing and formatting.

scale

public int getScale()

Returns the scale being used for numeric formatting. *getScale()* always returns -1 , meaning no scale is being used. Override this method if you want to use a different scale.

variantType

public abstract int getVariantType()

Returns the *Variant* type being used by this *Variant* *Formatter*. All calls to *getVariantType()* will produce *Variants* of this type. There is no *setVariantType()* because there is a special version of *parse()*, which allows the caller to request a particular returned *Variant* type.

VariantFormatter methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object

Method	Implemented in
format(com.borland.dx.dataset.Variant, com.borland.jb.util.FastStringBuffer)	this class
format(com.borland.dx.dataset.Variant)	this class
format(java.lang.Object)	this class
getSpecialObject(int)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
parse(com.borland.dx.dataset.Variant, char[], int, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant, int)	this class
parse(java.lang.String, com.borland.dx.dataset.Variant)	this class
parse(java.lang.String)	this class
setFromDouble(com.borland.dx.dataset.Variant, int, double)	this class
setFromInt(com.borland.dx.dataset.Variant, int, int)	this class
setPattern(java.lang.String)	this class
setSpecialObject(int, java.lang.Object)	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

format(com.borland.dx.dataset.Variant)

```
public abstract String format(Variant value)
```

Returns a *String* representing the given value stored in the supplied object. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

<i>value</i>	The value to be formatted to a <i>String</i> .
--------------	--

```
format(com.borland.dx.dataset.Variant,  
com.borland.jb.util.FastStringBuffer)
```

```
public FastStringBuffer format(Variant value, FastStringBuffer buffer)
```

Constructs a *FastStringBuffer* representing the given value stored in the supplied *Variant*. All reasonable attempts are made to “cast” the type found in the *Variant* into the appropriate type specified in the constructor of the

subclasses. A returned empty string indicates a **null** or empty input value. A **null** return means the formatting failed.

<i>value</i>	The value to be formatted. It will be cast to the appropriate type where possible (though the <i>Variant</i> itself will not be altered).
<i>buffer</i>	A <i>FastStringBuffer</i> that receives the formatted text. null is not permitted.

format(java.lang.Object)

public String format(Object value)

Returns a *String* representing the given value stored in the supplied object. All reasonable attempts are made to “cast” the type found in the object into the appropriate type specified in the constructor of the implementing classes. A returned empty string indicates a **null** or empty input value. **null** means the formatting failed.

<i>value</i>	The value to be formatted to a <i>String</i> .
--------------	--

Overrides com.borland.dx.text.ItemFormatter.format(java.lang.Object)

getSpecialObject(int)

public Object getSpecialObject(int objType)

Returns the value of the specified special object.

Some Formatter classes define special objects for their own use. You must know the internal details of the Format subclass being used to use *getSpecialObject()*.

<i>objType</i>	The special object type to return.
----------------	------------------------------------

Overrides com.borland.dx.text.ItemFormatter.getSpecialObject(int)

parse(com.borland.dx.dataset.Variant, char[], int, int)

public void parse(Variant variant, char[] value, int offset, int len)

A high-speed parse that parses directly into a character array.

<i>variant</i>	The parsed value (cannot be null).
<i>value</i>	The character array containing the text to parse.
<i>offset</i>	The zero-based offset into the character array.
<i>len</i>	The maximum number of characters in the array to use in the parse.

parse(java.lang.String)

```
public Object parse(String stringValue)
```

Analyzes the given *String* and produces as output an *Object* containing the appropriate value. A **null** return value results when *stringValue* is *null* or empty.

stringValue The string to be parsed.

Overrides com.borland.dx.text.ItemFormatter.parse(String)

parse(java.lang.String, com.borland.dx.dataset.Variant)

```
public abstract void parse(String stringValue, Variant value)
```

Analyzes the given *String* and produces as output a *Variant* containing the appropriate value.

stringValue The string to be parsed.

value The *Variant* that receives the parsed result.

parse(java.lang.String, com.borland.dx.dataset.Variant, int)

```
public void parse(String stringValue, Variant value, int variantType)
```

An alternative form of *parse()* that allows the type of *Variant* returned to be specified.

stringValue The string to be parsed. A **null** or empty *stringValue* returns a *VariantAssignedNull* variant.

value The *Variant* that receives the resulting data.

variantType The desired type of variant. If *variantType* is zero or one of the *VariantIsNull* types, the method chooses the default variant type specified at the time of the construction of *VariantFormatter*.

setFromDouble(com.borland.dx.dataset.Variant, int, double)

```
public static final void setFromDouble(Variant value, int type, double val)
```

Sets the variant using the specified **double** value.

value The variant value being set.

type The type of variant being set. Special handling occurs for the *Variant.FLOAT* type.

val The **double** value used for setting the variant.

setFromInt(com.borland.dx.dataset.Variant, int, int)

```
public static final void setFromInt(Variant value, int type, int val)
```

Sets the variant using the specified **int** value.

<i>value</i>	The variant value being set.
<i>type</i>	The type of variant being set. Special handling occurs for the <i>Variant.BYTE</i> and <i>Variant.SHORT</i> types.
<i>val</i>	The int value used for setting the variant.

setPattern(java.lang.String)

```
public String setPattern(String pattern)
```

Sets the pattern used for parsing and formatting to a new pattern, returning the old pattern. The new pattern must be of the same basic type associated with this type of formatter. For example, if you used a Date/Time pattern in the constructor, you can't switch to a numeric pattern as each basic pattern type has its own data-dependent *format()* and *parse()* methods.

If the new pattern is **null** (or empty), *setPattern()* chooses a default pattern for the current locale.

<i>pattern</i>	The new pattern to be used for formatting and parsing.
----------------	--

setSpecialObject(int, java.lang.Object)

```
public Object setSpecialObject(int charType, Object obj)
```

Some Formatter classes define special objects for their own use. This method allows them to be set. You must know the internal details of the Format subclass being used to use *setSpecialObject()*.

The returned value is the prior value of the object.

<i>charType</i>	The special object type. The possible values are <i>VariantFormatter.FillChar</i> , which is the fill character to fill blank slots, and <i>VariantFormatter.Replacecharacter</i> , which is used to replace <i>FillChar</i> on parse.
<i>obj</i>	The special object.

jb.io package

The *jb.io* package contains specialized classes for input and output. These classes are used by classes in other *com.borland* packages.

The following classes in this package are used internally. Do not use them directly:

- BufferedInputFile
- ByteToCharJava
- FileSystem
- LocalFileSystem
- TraverseAction

Interfaces

FileSystem

TraverseAction

Classes and components

AsciiInputStream

AsciiOutputStream

BufferedInputFile

ByteToCharJava

CharToByteJava

EncodedInputStrea

EncodedOutputStream

FastBufferedInputStream

FastBufferedOutputStream

InputStreamToByteArray

LocalFileSystem

SimpleCharInputStream

SimpleCharOutputStream

Overview of classes in the *com.borland.jb.io* package

<code>AsciiInputStream</code>	An implementation of <i>SimpleCharInputStream</i> that is optimized for data where most of the characters belong to either the ASCII character set or the 8859_1 character set. Other characters are assumed to be encoded in Unicode escapes.
<code>AsciiOutputStream</code>	An implementation of <i>SimpleCharOutputStream</i> that is optimized for data where most of the characters belong to either the ASCII character set or the 8859_1 character set. Other characters are assumed to be encoded in Unicode escapes.
<code>EncodedInputStream</code>	An implementation of <i>SimpleCharInputStream</i> that is optimized for data where most of the characters belong to the specified character set. Other characters are assumed to be encoded in Unicode escapes.
<code>EncodedOutputStream</code>	An implementation of <i>SimpleCharOutputStream</i> that is optimized for data where most of the characters belong to the specified character set. Other characters are assumed to be encoded in Unicode escapes.
<code>FastBufferedInputStream</code>	An unsynchronized buffered input stream to read in characters from a stream without causing a read every time.
<code>FastBufferedOutputStream</code>	An unsynchronized buffered output stream to read out characters from a stream without causing a read every time.
<code>SimpleCharInputStream</code>	This abstract class provides the capability of reading input one line at a time.
<code>SimpleCharOutputStream</code>	This abstract class provides the ability to write complete strings, arbitrarily-delimited strings, and terminated lines.
<code>InputStreamToByteArray</code>	A wrapper around <i>ByteArrayInputStream</i> .

AsciiInputStream class

jb.io package

Extends com.borland.jb.io.SimpleCharInputStream

An implementation of *SimpleCharInputStream* that is optimized for data where most of the characters belong to either the ASCII character set or the 8859_1 character set. Other characters are assumed to be encoded in Unicode escapes.

Note that under Western European versions of Windows, including the United States version, the javac compiler assumes the encoding of 8859_1, even though the actual encoding should be Cp1252. Cp1252 contains some characters that are not in 8859_1. If your source file contains these additional characters, they will not be correctly interpreted. In this case, you should specify Cp1252 as the encoding.

AsciiInputStream constructors

AsciiInputStream(java.io.InputStream)

```
public AsciiInputStream(InputStream in)
```

Creates a new buffered stream with a default buffer size of 2048 characters.

in The input stream.

AsciiInputStream(java.io.InputStream, int)

```
public AsciiInputStream(InputStream in, int size)
```

Creates a new buffered stream with the specified buffer size.

in The input stream.

size The buffer size.

AsciiInputStream properties

Property	Implemented in
class*	java.lang.Object

AsciiInputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	this class
readLine()	com.borland.jb.io.SimpleCharInputStream
toString()	java.lang.Object
unread(int)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

close()

public void close()

Closes the input stream. Should be the last operation done with this object.

Overrides com.borland.jb.io.SimpleCharInputStream.close()

read()

public int read()

Reads a byte of data. This method will block if no input is available. This method returns the byte read, or -1 if the end of the stream is reached. If an I/O error occurs, *read()* throws an *IOException*.

See also com.borland.jb.io.AsciiOutputStream.write(int)

Overrides com.borland.jb.io.SimpleCharInputStream.read()

unread(int)

public void unread(int undoChar)

“Pushes” the given character back into the input buffer so the next *read()* will return it.

undoChar The given character.

Overrides com.borland.jb.io.SimpleCharInputStream.unread(int)

AsciiOutputStream class

jb.io package

Extends com.borland.jb.io.SimpleCharOutputStream

An implementation of *SimpleCharOutputStream* that is optimized for data where most of the characters belong to either the ASCII character set or the 8859_1 character set. Other characters are assumed to be encoded in Unicode escapes.

Under Western European versions of Windows, including the United States version, the javac compiler assumes the encoding of 8859_1, even though the actual encoding should be Cp1252. Cp1252 contains some characters that are not in 8859_1. If your source file contains these additional characters, they will not be correctly interpreted. In this case, you should specify Cp1252 as the encoding.

AsciiOutputStream constructors

AsciiOutputStream(java.io.OutputStream)

public AsciiOutputStream(OutputStream out)

Creates a new buffered stream with a default buffer size of 2048 characters.

out The output stream.

AsciiOutputStream(java.io.OutputStream, int)

public AsciiOutputStream(OutputStream out, int bufferLength)

Creates a new buffered stream with the specified buffer size.

out The output stream.

bufferLength The buffer size.

AsciiOutputStream properties

Property	Implemented in
class*	java.lang.Object

AsciiOutputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
flush()	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(int)	this class
write(java.lang.String)	com.borland.jb.io.SimpleCharOutputStream
writeDelimited(java.lang.String, char)	com.borland.jb.io.SimpleCharOutputStream
writeln()	com.borland.jb.io.SimpleCharOutputStream
writeln(java.lang.String)	com.borland.jb.io.SimpleCharOutputStream

close()

public void close()

Closes the output stream. Should be the last operation done with this object.

Overrides com.borland.jb.io.SimpleCharOutputStream.close()

flush()

public void flush()

Causes all currently buffered information to be written to the output stream.

write(int)

public void write(int ch)

Writes the specified character. This method will block until the character is actually written.

If an I/O error occurs, *write(int)* throws an *IOException*.

ch The character to be written.

Overrides com.borland.jb.io.SimpleCharOutputStream.write(int)

EncodedInputStream class

jb.io package

Extends com.borland.jb.io.SimpleCharInputStream

An implementation of *SimpleCharInputStream* that is optimized for data where most of the characters belong to the specified character set. Other characters are assumed to be encoded in Unicode escapes.

EncodedInputStream constructors

EncodedInputStream(java.io.InputStream)

public EncodedInputStream(InputStream in)

Creates a new buffered stream with the default encoding and with a default buffer size of 2048 characters.

in The input stream.

EncodedInputStream(java.io.InputStream, java.lang.String)

public EncodedInputStream(InputStream in, String encoding)

Creates a new buffered stream with the specified encoding and with a default buffer size of 2048 characters.

in The input stream.

encoding The specified encoding.

EncodedInputStream(java.io.InputStream, java.lang.String, int)

public EncodedInputStream(InputStream in, String encoding, int size)

Creates a new buffered stream with the specified encoding and buffer size.

in The input stream.

encoding The specified encoding.

size The buffer size.

EncodedInputStream properties

Property	Implemented in
class*	java.lang.Object

EncodedInputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	this class
readLine()	com.borland.jb.io.SimpleCharInputStream
toString()	java.lang.Object
unread(int)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

close()

public void close()

Closes the encoded input stream. Should be the last operation done with this object.

Overrides com.borland.jb.io.SimpleCharInputStream.close()

read()

public int read()

Reads a byte of data. This method will block if no input is available. This method returns the byte read, or -1 if the end of the stream is reached. If an I/O error occurs, *read()* throws an *IOException*.

See also com.borland.jb.io.EncodedOutputStream.write(int)

Overrides com.borland.jb.io.SimpleCharInputStream.read()

unread(int)

public void unread(int undoChar)

“Pushes” the given character back into the input buffer so the next *read()* will return it.

undoChar The given character.

Overrides com.borland.jb.io.SimpleCharInputStream.unread(int)

EncodedOutputStream class

jb.io package

Extends com.borland.jb.io.SimpleCharOutputStream

An implementation of *SimpleCharOutputStream* that is optimized for data where most of the characters belong to the specified character set. Other characters are assumed to be encoded in Unicode escapes.

EncodedOutputStream constructors

EncodedOutputStream(java.io.OutputStream)

```
public EncodedOutputStream(OutputStream out)
```

Creates output stream encoded in the default encoding.

out The output stream.

EncodedOutputStream(java.io.OutputStream, java.lang.String)

```
public EncodedOutputStream(OutputStream out, String encodingString)
```

Creates output stream encoded in the specified encoding. Unicode characters passed in for a *write()* will be converted to native, potentially multibyte, forms. Characters that cannot be encoded in the specified encoding will be represented in Unicode escapes.

out The output stream.

encodingString The encoding in the resulting stream.

EncodedOutputStream(java.io.OutputStream, java.lang.String, int)

```
public EncodedOutputStream(OutputStream out, String encodingString, int bufferSize)
```

Creates output stream encoded in the specified encoding and the specified buffer size. Currently this class does not do its own buffering—the buffer size is ignored. Unicode characters passed in for a *write()* will be converted to native, potentially multibyte, forms.

out The output stream.

encodingString The encoding in the resulting stream.

EncodedOutputStream properties

Property	Implemented in
class*	java.lang.Object

EncodedOutputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
flush()	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(int)	this class
write(java.lang.String)	com.borland.jb.io.SimpleCharOutputStream
writeDelimited(java.lang.String, char)	com.borland.jb.io.SimpleCharOutputStream
writeln()	com.borland.jb.io.SimpleCharOutputStream
writeln(java.lang.String)	com.borland.jb.io.SimpleCharOutputStream

close()

public void close()

Closes the encoded output stream. Should be the last operation done with this object.

Overrides com.borland.jb.io.SimpleCharOutputStream.close()

flush()

public void flush()

Causes all currently buffered information to be written to the output stream.

write(int)

```
public void write(int ch)
```

Writes the character represented by the *ch* parameter. This method will block until the byte is actually written.

This method throws an *IOException* if an I/O error has occurred. The destination is encoded with the specified character set. Characters that cannot be encoded in the specified character set are encoded with Unicode escapes, like *'?'*. Malformed Unicode characters (characters that are invalid in any encoding) are converted to *'?'*.

ch The character to be written.

Overrides `com.borland.jb.io.SimpleCharOutputStream.write(int)`

FastBufferedInputStream class

jb.io package

Extends `java.io.FilterInputStream`

An unsynchronized buffered input stream that reads in characters from a stream without causing a read every time. The data is read into a buffer, then subsequent reads result in fast buffer access. This class is patterned after *java.io.BufferedInputStream*. The primary difference is that all access is unsynchronized (not thread-safe), for faster response.

FastBufferedInputStream variables

Variable	Defined in
<code>in</code>	<code>java.io.FilterInputStream</code>

FastBufferedInputStream constructors

FastBufferedInputStream(java.io.InputStream)

```
public FastBufferedInputStream(InputStream in)
```

Creates a new buffered stream with a default buffer size of 2048 characters.

in The input stream.

FastBufferedInputStream(java.io.InputStream, int)

public FastBufferedInputStream(InputStream in, int size)

Creates a new buffered stream with the specified buffer size.

- in* The in stream.
- size* The buffer size.

FastBufferedInputStream properties

Property	Implemented in
class*	java.lang.Object

FastBufferedInputStream methods

Method	Implemented in
available()	this class
clone()	java.lang.Object
close()	java.io.FilterInputStream
equals(java.lang.Object)	java.lang.Object
fill()	this class
finalize()	java.lang.Object
hashCode()	java.lang.Object
mark(int)	this class
markSupported()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	this class
read(byte[], int, int)	this class
read(byte[])	java.io.FilterInputStream
reset()	this class
skip(long)	this class
toString()	java.lang.Object
unread()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

available()

```
public int available()
```

Returns the number of bytes that can be read without blocking. This total is the number of bytes in the buffer and the number of bytes available from the input stream.

Overrides `java.io.FilterInputStream.available()`

fill()

```
protected void fill()
```

Reads as much as will fit into buffers.

mark(int)

```
public void mark(int readlimit)
```

Marks the current position in the input stream. A subsequent call to the *reset()* method will reposition the stream at the last marked position so that subsequent reads will re-read the same bytes. The stream promises to allow *readlimit* bytes to be read before the mark position gets invalidated.

readlimit The maximum limit of bytes allowed to be read before the mark position becomes invalid.

Overrides `java.io.FilterInputStream.mark(int)`

markSupported()

```
public boolean markSupported()
```

Returns a boolean indicating if this stream type supports mark/reset.

Overrides `java.io.FilterInputStream.markSupported()`

read()

```
public int read()
```

Reads a byte of data. This method will block if no input is available.

This method returns the byte read, or `-1` if the end of the stream is reached. If an I/O error occurs, *read()* throws an *IOException*.

Overrides `java.io.FilterInputStream.read()`

read(byte[], int, int)

```
public int read(byte[] copyBuffer, int off, int len)
```

Reads into an array of bytes. Blocks until some input is available.

This method returns the actual number of bytes read, or `-1` when the end of the stream is reached. If an I/O error occurs, this method throws an *IOException*.

copyBuffer The buffer into which the data is read.

len The maximum number of bytes read.

Overrides `java.io.FilterInputStream.read(byte[], int, int)`

reset()

`public synchronized void reset()`

Repositions the stream to the last marked position. If the stream has not been marked, or if the mark has been invalidated, an *IOException* is thrown.

Stream marks are intended to be used in situations where you need to read ahead a little to see what's in the stream. Often this is most easily done by invoking a general parser.

- If the stream is of the type handled by the parser, the operation will continue.
- If the stream is not of that type, the parser will generate an exception when it fails. If an exception gets thrown within *readlimit* bytes, the parser will allow the outer code to reset the stream and to try another parser.

Overrides `java.io.FilterInputStream.reset()`

skip(long)

`public long skip(long n)`

Skips the specified number of bytes of input.

This method returns the actual number of bytes skipped. If an I/O error occurs, it throws an *IOException*.

n The number of bytes to be skipped.

Overrides `java.io.FilterInputStream.skip(long)`

unread()

`public void unread()`

"Pushes" the given character back into the input buffer so the next *read()* will return it.

FastBufferedOutputStream class

jb.io package

Extends java.io.FilterOutputStream

An unsynchronized buffered output stream that reads out characters from a stream without causing a read every time. The data is read from a buffer, then subsequent reads result in fast buffer access. This class is patterned after *java.io.BufferedOutputStream*. The primary difference is that all access is unsynchronized (not thread-safe), for faster response.

FastBufferedOutputStream variables

Variable	Defined in
buf	this class
count	this class
out	java.io.FilterOutputStream

buf

protected byte[] buf

The buffer where data is stored.

count

protected int count

The number of bytes in the buffer.

FastBufferedOutputStream constructors

FastBufferedOutputStream(java.io.OutputStream)

public FastBufferedOutputStream(OutputStream out)

Creates a new buffered stream with a default buffer size of 2048 characters.

out The output stream.

FastBufferedOutputStream(java.io.OutputStream, int)

```
public FastBufferedOutputStream(OutputStream out, int size)
```

Creates a new buffered stream with the specified buffer size.

out The output stream.

<i>size</i>	The buffer size.
-------------	------------------

FastBufferedOutputStream properties

Property	Implemented in
class*	java.lang.Object

FastBufferedOutputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	java.io.FilterOutputStream
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
flush()	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(byte[], int, int)	this class
write(byte[])	java.io.FilterOutputStream
write(int)	this class

flush()

```
public void flush()
```

Flushes the stream. This will write any buffered output bytes.

This method throws an *IOException* if an I/O error occurs.

Overrides `java.io.FilterOutputStream.flush()`

write(byte[], int, int)

```
public void write(byte[] b, int off, int len)
```

Writes a subarray of bytes. This method throws an *IOException* if an I/O error occurs.

b The data to be written.

off The start offset in the data.

len The number of bytes that are written.

Overrides `java.io.FilterOutputStream.write(byte[], int, int)`

write(int)

```
public void write(int b)
```

Writes a byte. This method will block until the byte is actually written.

This method throws an *IOException* if an I/O error occurs.

b The byte to be written.

Overrides `java.io.FilterOutputStream.write(int)`

InputStreamToByteArray class

jb.io package

Extends `java.io.ByteArrayInputStream`

This class is a wrapper around *java.io.ByteArrayInputStream*.

InputStreamToByteArray variables

Variable	Defined in
<code>buf</code>	<code>java.io.ByteArrayInputStream</code>
<code>count</code>	<code>java.io.ByteArrayInputStream</code>
<code>mark</code>	<code>java.io.ByteArrayInputStream</code>
<code>pos</code>	<code>java.io.ByteArrayInputStream</code>

InputStreamToByteArray constructors

InputStreamToByteArray(byte[])

public InputStreamToByteArray(byte[] buf)

Constructs an *InputStreamToByteArray* object with the following parameters:

buf The buffer into which the data is read.

InputStreamToByteArray(byte[], int, int)

public InputStreamToByteArray(byte[] buf, int offset, int length)

Constructs an *InputStreamToByteArray* object with the following parameters:

buf The buffer into which the data is read.

offset The start offset of the data.

length The maximum number of bytes read.

InputStreamToByteArray properties

Property	Implemented in
bytes*	this class
class*	java.lang.Object

bytes

public byte[] getBytes()

The number of bytes read.

InputStreamToByteArray methods

Method	Implemented in
available()	java.io.ByteArrayInputStream
clone()	java.lang.Object
close()	java.io.ByteArrayInputStream
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getBytes(java.io.InputStream)	this class
hashCode()	java.lang.Object
mark(int)	java.io.ByteArrayInputStream

Method	Implemented in
markSupported()	java.io.ByteArrayInputStream
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	java.io.ByteArrayInputStream
read(byte[], int, int)	java.io.ByteArrayInputStream
read(byte[])	java.io.InputStream
reset()	java.io.ByteArrayInputStream
skip(long)	java.io.ByteArrayInputStream
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

getBytes(java.io.InputStream)

public static byte[] getBytes(InputStream stream)

A static method that returns an array of bytes representing the *InputStream* with the specified *stream*.

stream The specified stream.

SimpleCharInputStream class (abstract)

jb.io package

Extends java.lang.Object

Extended by com.borland.jb.io.AsciiInputStream, com.borland.jb.io.EncodedInputStream

This abstract class provides subclasses with the capability of reading input one line at a time. This class is used by *AsciiEncodedInputStream* and *AsciiEncodedOutputStream* classes.

SimpleCharInputStream properties

Property	Implemented in
class*	java.lang.Object

SimpleCharInputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
read()	this class
readLine()	this class
toString()	java.lang.Object
unread(int)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

close()

public abstract void close()

Closes the input stream. Should be the last operation done with this object. On error, this method throws an *IOException*.

read()

public abstract int read()

Reads a byte of data. This method will block if no input is available. This method returns the byte read, or -1 if the end of the stream is reached. If an I/O error occurs, this method throws an *IOException*.

readLine()

public String readLine()

Reads a complete line of text, which is defined as a line terminated with a line-feed character or with a carriage-return/linefeed pair. Carriage returns that are not immediately followed by a linefeed character are not treated as line terminators but as part of the current line. On error this method throws an *IOException*.

unread(int)

public abstract void unread(int ch)

“Pushes” the given character back into the input buffer so the next *read()* will return it. On error this method throws an *IOException*.

ch The given character.

SimpleCharOutputStream class (abstract)

jb.io package

Extends java.lang.Object

Extended by com.borland.jb.io.AsciiOutputStream,
com.borland.jb.io.EncodedOutputStream

This abstract class provides subclasses with the ability to write complete strings, arbitrarily-delimited strings, and terminated lines.

SimpleCharOutputStream properties

Property	Implemented in
class*	java.lang.Object

SimpleCharOutputStream methods

Method	Implemented in
clone()	java.lang.Object
close()	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object
write(int)	this class
write(java.lang.String)	this class
writeDelimited(java.lang.String, char)	this class

Method	Implemented in
<code>writeln()</code>	this class
<code>writeln(java.lang.String)</code>	this class

close()

public abstract void close()

Closes the output stream. Should be the last operation done with this object. On error, this method throws an *IOException*.

write(int)

public abstract void write(int ch)

Writes the value represented by *ch* to the output stream. On error, this method throws an *IOException*.

ch The value to write to the output stream.

write(java.lang.String)

public void write(String string)

Writes all the characters in the given string to the output stream. On error, this method throws an *IOException*.

string The string to write to the output stream.

writeDelimited(java.lang.String, char)

public void writeDelimited(String string, char delimiter)

Writes a string with the given delimiter. Each occurrence of the given delimiter is escaped by repeating the delimiter. On error, this method throws an *IOException*.

string The string to write to the output stream.

delimiter The delimiter to follow the *string*.

writeln()

public void writeln()

Writes out a line delimiter. The line delimiter in this case is a fixed line-feed character, regardless of the line-terminating convention of the current environment. On error, this method throws an *IOException*.

println(java.lang.String)

public void println(String string)

Writes the given string and follows it with a line feed. The line delimiter in this case is a fixed line-feed character, regardless of the line-terminating convention of the current environment. On error, this method throws an *IOException*.

string The string to write to the output stream.

jb.util package

The *jb.util* package contains utility classes and interfaces. These classes are used by classes in other *com.borland* packages.

The *jb.util* package contains the following types of classes:

- Diagnostic classes
- Event-dispatching classes
- Exception-related classes
- I/O classes
- Internationalization classes
- Multicaster classes
- TriState interfaces
- Miscellaneous util classes

Trace is an internal use only class. Do not use this class directly.

Interfaces

ChainedException
Trace

ExceptionDispatch
TriStateProperty

ExceptionHandler
VetoableDispatch

Classes and components

ArrayResourceBundle
DispatchableEvent
ExceptionChain
LocaleUtil

BasicBeanInfo
ErrorResponse
FastStringBuffer
SearchPath

Diagnostic
EventMulticaster
Hex
VetoException

Overview of classes in the *com.borland.jb.util* package

Diagnostic classes

Diagnostic	A collection of diagnostic functions for debugging program flow and output.
------------	---

Event-dispatching classes

DispatchableEvent	An interface for dispatchable events.
VetoableDispatch	An interface for events that can be vetoed.
VetoException	Thrown when an event listener wants to halt multicasting of a vetoable event.

Exception-related classes

ErrorResponse	Used for collecting a response to an error from a component.
ExceptionDispatch	An interface for events that can throw exceptions that <i>EventMulticaster</i> can send to multiple listeners.
ExceptionChain	Represents the node of a <i>ChainedException</i> object.
ExceptionHandler	An interface that allows an object to generically handle exceptions.
ChainedException	An interface that collects generic routines to handle chained exceptions.

I/O classes

FastStringBuffer	A class to use instead of <i>StringBuffer</i> when a buffer is not shared. Avoids complications of synchronization and sharing.
------------------	---

Internationalization classes

ArrayResourceBundle	An abstract subclass of <i>java.util.ResourceBundle</i> that manages locale-dependent resources in an array. Uses numeric references rather than string references, for less overhead and better performance than JDK resource bundle classes.
LocaleUtil	Returns the locale identified by a single string with “locale_country_variant” as returned by <i>Locale.toString()</i> .

Multicaster classes

EventMulticaster	For user-defined events. Used for model events and selection events.
------------------	--

TriState interfaces

TriStateProperty	A general-purpose interface containing the values TRUE, FALSE, and DEFAULT.
------------------	---

Miscellaneous util classes

BasicBeanInfo	A convenient implementation of the BeanInfo interface, designed to be subclassed to fill in appropriate properties, methods, and events for a JavaBean.
Hex	Static definitions of hexadecimal characters as chars and as bytes.
SearchPath	Encapsulates a search path (such as classpath), and can perform searches along that path given a relative directory and filename.

ArrayResourceBundle class (abstract)

jb.util package

Extends java.util.ResourceBundle

The *ArrayResourceBundle* is an abstract subclass of *java.util.ResourceBundle* that manages locale-dependent resources in an array. By using numeric references rather than string references, it requires less overhead and

provides better performance than *java.util.PropertyResourceBundle* and *java.util.ListResourceBundle*.

Subclasses must override the *getContents()* method and provide an array, where each item in the array is the resource value. The key for each resource value is its numeric offset in the array. For example, the first element in the array has the key 0. It may be retrieved by using either *getObject(0)* or *getObject("0")*.

Unlike *ListResourceBundle* and *PropertyResourceBundle*, where each locale-specific variation of a bundle can override only selected resources, each variation of *ArrayResourceBundle* must provide the complete set of resources. For example, if the custom class *MyResources* has three resources, then its subclasses *MyResources_ja* and *MyResources_fr* must also have three resources.

The following example shows the structure of a *ResourceBundle* based on *ArrayResourceBundle*.

```
class MyResource extends ResourceBundle {
    public Object []getContents() {
        return contents;
    }
    static final Object []contents = {
        // LOCALIZE THIS
        "Yes",    // Label for the YES button
        "No",     // Label for the NO button
        "Cancel"  // Label for the CANCEL button
        // END OF MATERIAL TO LOCALIZE
    };
}
```

ArrayResourceBundle variables

Variable	Defined in
parent	java.util.ResourceBundle

ArrayResourceBundle properties

Property	Implemented in
class*	java.lang.Object
keys*	this class
locale*	java.util.ResourceBundle

keys

public Enumeration getKeys()

Returns an enumeration of the keys.

Overrides *java.util.ResourceBundle.getKeys()*

ArrayResourceBundle methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getBundle(java.lang.String, java.util.Locale, java.lang.ClassLoader)	java.util.ResourceBundle
getBundle(java.lang.String, java.util.Locale)	java.util.ResourceBundle
getBundle(java.lang.String)	java.util.ResourceBundle
getContents()	this class
getObject(int)	this class
getObject(java.lang.String)	java.util.ResourceBundle
getString(int)	this class
getString(java.lang.String)	java.util.ResourceBundle
getStringArray(int)	this class
getStringArray(java.lang.String)	java.util.ResourceBundle
handleGetObject(java.lang.String)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
setParent(java.util.ResourceBundle)	java.util.ResourceBundle
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

getContents()

protected abstract Object[] getContents()

Gets the contents of the array. See “About the ArrayResourceBundle class” for more information.

getObject(int)

```
public Object getObject(int index)
```

Gets an element in the array. If *index* is 0, the first element in the array is retrieved.

index The element in the array to retrieve.

getString(int)

```
public final String getString(int key)
```

Gets an object from an *ArrayResourceBundle*. This is a convenience method that saves the extra step of casting by returning a *String*. If an error occurs, *getString(int)* throws a *MissingResourceException*.

key The numeric offset of the resource value in the array.

getStringArray(int)

```
public final String[] getStringArray(int key)
```

Gets an object from a *ResourceBundle*. This is a convenience method that saves the extra step of casting by returning a *String*. If an error occurs, *getStringArray(int)* throws a *MissingResourceException*.

key The numeric offset of the resource value in the array.

handleGetObject(java.lang.String)

```
protected Object handleGetObject(String key)
```

Gets an object from a *ResourceBundle*. If the specified key is not found, *handleGetObject* must return null.

key The numeric offset of the resource value in the array.

Overrides `java.util.ResourceBundle.handleGetObject(java.lang.String)`

BasicBeanInfo class (abstract)

jb.util package

Extends `java.lang.Object`

Extended by `com.borland.datastore.DataStoreBeanInfo,`
 `com.borland.datastore.DataStoreConnectionBeanInfo,`
 `com.borland.datastore.TxManagerBeanInfo,`
 `com.borland.dx.dataset.ColumnBeanInfo,`

com.borland.dx.dataset.DataSetViewBeanInfo,
 com.borland.dx.dataset.ParameterRowBeanInfo,
 com.borland.dx.dataset.StorageDataSetBeanInfo,
 com.borland.dx.dataset.TextDataFileBeanInfo,
 com.borland.dx.sql.dataset.DatabaseBeanInfo,
 com.borland.dx.sql.dataset.ProcedureResolverBeanInfo,
 com.borland.dx.sql.dataset.QueryResolverBeanInfo

Implements java.beans.BeanInfo

A convenient implementation of the *java.beans.BeanInfo* interface, designed to be subclassed to fill in appropriate properties, methods, and events for a JavaBean. Extend *BasicBeanInfo* when you want to provide explicit information about your component rather than have JBuilder and other such tools derive the information through introspection.

BasicBeanInfo variables

Variable	Defined in
additionalBeanInfo	this class
beanClass	this class
CONTAINER_DELEGATE	this class
customizerClass	this class
defaultEventIndex	this class
defaultPropertyIndex	this class
ENUMERATION	this class
eventListenerMethods	this class
eventSetDescriptors	this class
iconColor16x16	this class
iconColor32x32	this class
iconMono16x16	this class
iconMono32x32	this class
IS_CONTAINER	this class
LATE_SETTING	this class
methodNames	this class
methodParameters	this class
namedAttributes	this class
propertyDescriptorAttributes	this class
propertyDescriptors	this class

additionalBeanInfo

protected BeanInfo[] additionalBeanInfo

An array of other other bean information objects.

beanClass

protected Class beanClass

The JavaBean component class. A subclassed bean information class must specify a component class, which is the only required field.

CONTAINER_DELEGATE

public static final String CONTAINER_DELEGATE = "containerDelegate"

A Bean descriptor key value used to inform a designer that the add calls and layout setting should not be applied directly to the component, but should call this method first.

The method is assumed to take no parameters. *javax.swing.JFrame* contains the following example:

```
setValue("containerDelegate", "getContentPane");
```

customizerClass

protected Class customizerClass

The customizer class for this JavaBean, if one exists.

defaultEventIndex

protected int defaultEventIndex

The index of the default event for your JavaBean. The index identifies the event in the set of event descriptors described held in the array of the the *eventSetDescriptors* property. An index value of -1 means there is no default event.

defaultPropertyIndex

protected int defaultPropertyIndex

The index of the default property for your JavaBean. The index identifies the property in the set of property descriptors held in the array of the the *propertyDescriptors* property. An index value of -1 means there is no default property.

ENUMERATION

public static final String ENUMERATION = "enumerationValues"

A property descriptor key value that will cause a tag list property editor to be associated with this property.

The value for this key should be an *Object* array with three values for each entry desired in the tag list. The first value in each set is the text that will

appear in the tag list, the next value is the live value and the third value is the java Initialization String.

eventListenerMethods

protected String[][] eventListenerMethods

The names of each event set's listener methods. Specify the names using this format:

```
{{"listener1Method1", "listener1Method2",
  "listener1Method3", ...}, ...}
```

Example {"actionPerformed"}, ...}

eventSetDescriptors

protected String[][] eventSetDescriptors

The event information for your JavaBean. Specify the event information using this format:

```
{"EventSetName", "EventListenerClass", "AddMethod", "RemoveMethod"}, ...}
```

Example {"ActionListener", "java.awt.event.ActionListener", "addActionListener",
 "removeActionListener"}, ...}

iconColor16x16

protected Image iconColor16x16

A 16x16 pixel color icon for your JavaBean.

iconColor32x32

protected Image iconColor32x32

A 32x32 pixel color icon for your JavaBean.

iconMono16x16

protected Image iconMono16x16

A 16x16 pixel monochromatic icon for your JavaBean.

iconMono32x32

protected Image iconMono32x32

A 32x32 pixel monochromatic icon for your JavaBean.

IS_CONTAINER

```
public static final String IS_CONTAINER = "isContainer"
```

A Bean descriptor key value used to inform a designer that although the bean extends *java.awt.Container* it should not be treated as one. The value should be **false**.

When not present, beans that extend *java.awt.Container* are treated as a containers.

LATE_SETTING

```
public static final String LATE_SETTING = "lateSetting"
```

A property descriptor key value that operates as a hint to the designer that this property setting should come near the end (after the add calls). The value should be set to **true**.

When not present, this variable is assumed to be **false**.

methodNames

```
protected String[] methodNames
```

The method names for your JavaBean. Don't include the access methods for properties. Specify the method names using this format:

```
{ "method1", "method2", "method3", ... }
```

Example

```
{ "fillRect", "eraseRect", "close", "open" }
```

methodParameters

```
protected String[][] methodParameters
```

The method parameters for each of your JavaBean's methods. Specify the parameters using this format:

```
{{ "method1Parameter1", "method1Parameter2", ... }, ... }
```

Example

```
{{ "java.awt.Graphics", "java.awt.Rectangle", ... }, ... }
```

namedAttributes

```
protected Object[][] namedAttributes
```

Any additional named attributes for the JavaBean. Specify the attributes using this format:

```
{{ "AttributeName", AttributeSetting }, ... }
```

Example

```
{{ "isContainer", Boolean.TRUE }, { "containerDelegate", "getContentPane", ... }
```

propertyDescriptorAttributes

protected Object[][] propertyDescriptorAttributes

Additional attributes for each property described in the *propertyDescriptors* array. Entries in the *propertyDescriptorAttributes* array and the *propertyDescriptors* array are matched by an index position in each array. Use a **null** value as a placeholder for property entries without attributes. The *propertyDescriptorAttributes* array need not be the same size as the *propertyDescriptors* array.

Specify the attributes using this format:

```
{{"AttributeName", "AttributeValueAsString"}, ...}
```

Example

```
{null, {"enumerationValues", "LEFT, 2, SwingConstants.LEFT, CENTER, 0, SwingConstants.CENTER"},}
```

propertyDescriptors

protected String[][] propertyDescriptors

The property information for your JavaBean. Specify the property information using this format:

```
{{"PropertyName", "ReadMethod", "WriteMethod",  
  "PropertyEditor"}, ...}
```

Example

```
{{"text", "getText", "setText", null}, ...}
```

BasicBeanInfo properties

Property	Implemented in
additionalBeanInfo*	this class
beanDescriptor*	this class
class*	java.lang.Object
defaultEventIndex*	this class
defaultPropertyIndex*	this class
eventSetDescriptors*	this class
methodDescriptors*	this class
propertyDescriptors*	this class

additionalBeanInfo

public BeanInfo[] getAdditionalBeanInfo()

Returns the array of bean information objects for this JavaBean.

beanDescriptor

public BeanDescriptor getBeanDescriptor()

Returns the bean descriptor associated with this JavaBean.

defaultEventIndex

public int getDefaultEventIndex()

Returns the default event index for this JavaBean. The index returned locates the event in the eventSetDescriptors array. An index value of -1 means there is no default event for this bean.

defaultPropertyIndex

public int getDefaultPropertyIndex()

Returns the default property index for this JavaBean. The index returned locates the property in the propertyDescriptors array. An index value of -1 means there is no default property for this bean.

eventSetDescriptors

public EventSetDescriptor[] getEventSetDescriptors()

Returns the array of event set descriptors for this JavaBean.

methodDescriptors

public MethodDescriptor[] getMethodDescriptors()

Returns the array of method descriptors for this JavaBean.

propertyDescriptors

public PropertyDescriptor[] getPropertyDescriptors()

Returns the array of property descriptors for this JavaBean.

BasicBeanInfo methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getDefaultIconResource(int)	this class
getIcon(int)	this class
getImage(java.lang.String)	this class

Method	Implemented in
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

getDefaultIconResource(int)

protected String getDefaultIconResource(int iconKind)

Returns the default resource to use to find an icon for a JavaBean.

If an icon is requested, and not explicitly set in the subclass of *BasicBeanInfo*, an icon is searched for using a simple look-up using the *beanClass*'s name and the requested icon type.

Following is a simple chart describing the default resource locations using an example of the JavaBean class *package1.package2.MyBean*:

iconKind The **int** representing the icon type from *java.beans.BeanInfo*

getImage(java.lang.String)

public Image getImage(String resource)

This is a simple utility function that retrieves an *Image* object from a resource. The resource must be specified as a relative path to the *beanClass* resource.

resource The resource name relative to the *beanClass* resource.

ChainedException interface

jb.util package

Implemented by com.borland.datastore.DataStoreException,
com.borland.datastore.TxException,
com.borland.dx.dataset.DataSetException,
com.borland.dx.dataset.ValidationException,
com.borland.dx.sql.dataset.ResolutionException

The *ChainedException* interface collects generic routines to handle chained exceptions (a linked list of exceptions). In the *dataset* package, the *DataSetException* class handles exceptions in this manner. It can contain other types of *Exception* objects including those of type *ValidationException*,

InvalidFormatException, *ResolutionException* as well as other exceptions such as JDBC and SQL exceptions.

See also *com.borland.dx.dataset.DataSetException*

ChainedException properties

Property	Implemented in
exceptionChain*	this class

exceptionChain

public ExceptionChain getExceptionChain()

Returns to the beginning of the exception chain.

Diagnostic class

jb.util package

Extends java.lang.Object

The *Diagnostic* component collects useful diagnostic functions for debugging program flow and output.

All calls to *Diagnostic* methods with a **void** return type can be removed from the compiled classes by using the compiler's exclude class option:

```
-exclude com.borland.jb.util.Diagnostic
```

Diagnostic variables

Variable	Defined in
count	this class
out	this class
outputEnabled	this class

count

public static int count = 0

A common counter variable used for line numbers for debug output messages. For example,

```
Diagnostic.out.println(++Diagnostic.count+"\tdebug message");
```

out

```
public static PrintStream out = System.err
```

Enables or disables output of diagnostic messages to `System.err`.

outputEnabled

```
public static boolean outputEnabled = System.getProperty("jb.util.diagnostic", "on").equals("on")
```

Specifies whether output logging is initially enabled or disabled.

Diagnostic properties

Property	Implemented in
class*	java.lang.Object

Diagnostic methods

Method	Implemented in
addTraceCategory(java.lang.Object)	this class
check(boolean, java.lang.Object)	this class
check(boolean)	this class
clone()	java.lang.Object
enableChecking(boolean)	this class
enableOutput(boolean)	this class
equals(java.lang.Object)	java.lang.Object
exit(int)	this class
fail()	this class
fail(java.lang.Exception)	this class
fail(java.lang.Object)	this class
finalize()	java.lang.Object
flush()	this class
getTraceLevel()	this class
hashCode()	java.lang.Object
needException()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
precondition(boolean, java.lang.String)	this class
precondition(boolean)	this class
print(java.lang.String)	this class
println(java.lang.String)	this class
printlnc(java.lang.String)	this class

Method	Implemented in
<code>printStackTrace()</code>	this class
<code>printStackTrace(java.lang.Throwable)</code>	this class
<code>removeTraceCategory(java.lang.Object)</code>	this class
<code>setLogStream(java.io.PrintStream)</code>	this class
<code>setTraceLevel(int)</code>	this class
<code>toString()</code>	<code>java.lang.Object</code>
<code>trace(int, java.lang.String)</code>	this class
<code>trace(java.lang.Object, int, java.lang.String)</code>	this class
<code>trace(java.lang.Object, java.lang.String)</code>	this class
<code>wait()</code>	<code>java.lang.Object</code>
<code>wait(long, int)</code>	<code>java.lang.Object</code>
<code>wait(long)</code>	<code>java.lang.Object</code>
<code>warn(int, boolean, java.lang.String)</code>	this class
<code>warn(java.lang.Object, boolean, java.lang.String)</code>	this class
<code>warn(java.lang.Object, int, boolean, java.lang.String)</code>	this class
<code>warn(java.lang.Object, java.lang.String)</code>	this class

addTraceCategory(java.lang.Object)

`public static void addTraceCategory(Object category)`

A category-based tracing or warning method.

To set up a category-based tracing or warning, pass in a unique *String*, *Class* or other object that supports a meaningful *toString* operation. When a call to a trace or warn method that takes a category is made (for example, *trace(Object category, String description)*), the trace is displayed if an *addTraceCategory()* call was made with the same category object.

category The category object.

See also *removeTraceCategory(java.lang.Object)*

check(boolean)

`public static void check(boolean condition)`

Checks a condition within a method body.

An *IllegalStateException* is thrown if the given condition is **false**.

condition The boolean condition.

check(boolean, java.lang.Object)

public static void check(boolean condition, Object description)

Checks a condition within a method body.

Use this method to describe assumed results and state after internal operations.

A check is raised if the given condition is not true. An error here usually indicates an internal problem with the class.

enableChecking(boolean)

public static void enableChecking(boolean enable)

Enables or disables the checking of conditions in *precondition()* and *check()*.

See also *check(boolean)*, *check(boolean, java.lang.String)*, *precondition(boolean)*, *precondition(boolean, java.lang.String)*

enableOutput(boolean)

public static void enableOutput(boolean enable)

Enables or disables all output of diagnostic messages to `System.err`.

exit(int)

public static void exit(int code)

Calls `System.exit(int code)`.

code The method body to exit.

fail()

public static void fail()

Calls *check(false)* to force a failure.

fail(java.lang.Exception)

public static void fail(Exception ex)

Calls *check(false)* to force a failure but prints the exception message on the stack trace first.

fail(java.lang.Object)

public static void fail(Object description)

Causes a check exception if the code reaches an unexpected location.

flush()

public static void flush()

Flushes the diagnostic out *Stream* buffer.

getTraceLevel()

public static int getTraceLevel()

Gets the minimum threshold for trace and warning output. 0 is highest level and `+maxint` is lowest level.

See also *setTraceLevel(int)*

needException()

public static void needException()

Used to mark places where an *Exception* is needed.

Upon error, throws an *IllegalStateException*.

precondition(boolean)

public static void precondition(boolean condition)

Checks a condition. Same as *check(boolean)*, but typically placed at the start of the method body.

An *IllegalStateException* is thrown if the given condition is **false**.

condition The boolean condition, either **true** or **false**.

precondition(boolean, java.lang.String)

public static void precondition(boolean condition, String description)

Checks a condition. Same as *check(boolean, java.lang.String)*, but typically placed at the start of the method body.

An *IllegalStateException* is thrown if the given condition is **false**.

condition The boolean condition, either **true** or **false**.

description The description to print.

print(java.lang.String)

public static void print(String message)

Prints a message to the diagnostic out stream.

message The message to print.

println(java.lang.String)

public static void println(String message)

Prints a message to the diagnostic out stream, preceded by a line number (incremented count).

message The message to print.

printlnc(java.lang.String)

public static void printlnc(String message)

Prints a message to the diagnostic out stream, preceded by a line number (incremented count) and a tab character.

message The message to print.

printStackTrace()

public static void printStackTrace()

Prints a diagnostic stack trace of the current thread to the diagnostic out stream.

printStackTrace(java.lang.Throwable)

public static void printStackTrace(Throwable ex)

Prints a diagnostic stack trace of the current thread to the diagnostic out stream. Throws an exception.

removeTraceCategory(java.lang.Object)

public static void removeTraceCategory(Object category)

Removes a trace added with *addTraceCategory()*.

category The category object.

See also *addTraceCategory(java.lang.Object)*

setLogStream(java.io.PrintStream)

public static void setLogStream(PrintStream log)

Explicitly sets the stream for diagnostic messages to be sent to.

log The stream for messages to be sent to.

setTraceLevel(int)

```
public static void setTraceLevel(int level)
```

Sets the minimum threshold for trace and warning output.

Note This does not affect direct access to ‘out’, nor does it affect output in the checking functions. Use *enableOutput()* instead.

level The trace level. 0 is highest level and +maxint is lowest level. Setting this level to -1 effectively turns off traces and warnings.

See also *getTraceLevel()*

trace(int, java.lang.String)

```
public static void trace(int level, String description)
```

Outputs a trace if the threshold level is high enough and general output is enabled.

level The trace level. 0 is highest level and +maxint is lowest level. Setting this level to -1 effectively turns off traces and warnings.

description The string to trace.

trace(java.lang.Object, int, java.lang.String)

```
public static void trace(Object category, int level, String description)
```

Outputs a trace if the category and general output are both enabled, and the threshold level is high enough.

category The category object to trace.

level The trace level. 0 is highest level and +maxint is lowest level. Setting this level to -1 effectively turns off traces and warnings.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

trace(java.lang.Object, java.lang.String)

```
public static void trace(Object category, String description)
```

Outputs a trace if the category and general output are enabled.

category The category object to trace.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

warn(int, boolean, java.lang.String)

public static void warn(int level, boolean condition, String description)

Outputs a warning if the threshold level is high enough, the boolean condition is **true**, and general output is enabled.

level The trace level. 0 is highest level and +maxint is lowest level. Setting this level to -1 effectively turns off traces and warnings.

condition The boolean condition, either **true** or **false**.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

warn(java.lang.Object, boolean, java.lang.String)

public static void warn(Object category, boolean condition, String description)

Outputs a warning if the category object is enabled, the boolean condition is **true**, and general output is enabled.

category The category object to trace.

condition The boolean condition, either **true** or **false**.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

warn(java.lang.Object, int, boolean, java.lang.String)

public static void warn(Object category, int level, boolean condition, String description)

Outputs a warning if the category object is enabled, the boolean condition is **true**, and general output is enabled.

category The category object to trace.

level The trace level. 0 is highest level and +maxint is lowest level. Setting this level to -1 effectively turns off traces and warnings.

condition The boolean condition, either **true** or **false**.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

warn(java.lang.Object, java.lang.String)

public static void warn(Object category, String description)

Outputs a warning if the category and general output are both enabled.

category The category object to trace.

description The string to trace.

See also *addTraceCategory(java.lang.Object)*

DispatchableEvent class (abstract)

jb.util package

Extends java.util.EventObject

Extended by com.borland.datastore.jdbc.ServerStatusEvent,
com.borland.dbswing.DBRuntimeEvent,
com.borland.dx.dataset.AccessEvent,
com.borland.dx.dataset.DataChangeEvent,
com.borland.dx.dataset.ExceptionEvent, com.borland.dx.dataset.LoadEvent,
com.borland.dx.dataset.MasterNavigateEvent,
com.borland.dx.dataset.NavigationEvent,
com.borland.dx.dataset.ResponseEvent, com.borland.dx.dataset.StatusEvent

Implements java.io.Serializable

An abstract base class for dispatchable events that *com.borland.jb.util.EventMulticaster* can send to multiple listeners. An *EventMulticaster* can also send *VetoableDispatch* and *ExceptionDispatch* events, but these are not dispatchable events.

DispatchableEvent variables

Variable	Defined in
source	java.util.EventObject

DispatchableEvent constructors

DispatchableEvent(java.lang.Object)

public DispatchableEvent(Object source)

Constructs a *DispatchableEvent* event.

source The *Object* that generates the event.

DispatchableEvent properties

Property	Implemented in
class*	java.lang.Object
exceptionChain*	this class
source*	java.util.EventObject

exceptionChain

public ExceptionChain getExceptionChain()

Returns the exception chain for this event.

DispatchableEvent methods

Method	Implemented in
appendException(java.lang.Exception)	this class
clone()	java.lang.Object
dispatch(java.util.EventListener)	this class
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
paramString()	this class
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

appendException(java.lang.Exception)

public void appendException(Exception ex)

Adds the specified exception to the end of the exception chain for this event.

ex An exception that occurred.

dispatch(java.util.EventListener)

public abstract void dispatch(EventListener listener)

An abstract dispatch method.

listener The listener the event is sent to.

paramString()

protected String paramString()

Returns an empty string: “ ”. The string is the parameter string for the exception.

toString()

public String toString()

Converts the exception to a string that contains the name of the exception and its parameter string.

Overrides java.util.EventObject.toString()

ErrorResponse component

jb.util package

Extends java.lang.Object

Extended by com.borland.dx.dataset.ResolverResponse

The *ErrorResponse* component is used to collect a response to an error from an event listener. It contains three constants that provide possible responses: abort, ignore, and retry. It also has three methods that return one of the three responses, three **boolean** properties that test for the responses, and a *response* property that holds the response value.

ErrorResponse variables

Variable	Defined in
ABORT	this class
IGNORE	this class
response	this class
RETRY	this class

ABORT

public static final int ABORT = 1

The response is abort.

IGNORE

public static final int IGNORE = 2

The response is ignore.

response

protected int response

The response:

- 1 = ABORT
- 2 = IGNORE
- 3 = RETRY

RETRY

public static final int RETRY = 3

The response is retry.

ErrorResponse constructors

ErrorResponse()

public ErrorResponse()

Constructs an *ErrorResponse* object with a default response of ABORT.

ErrorResponse properties

Property	Implemented in
abort*	this class
class*	java.lang.Object
ignore*	this class
response*	this class
retry*	this class

abort

public final boolean isAbort()

Tests to see if the response is ABORT. If it returns **true**, the response is ABORT.

ignore

public final boolean isIgnore()

Tests to see if the response is IGNORE. If it returns **true**, the response is IGNORE.

response

public final int getResponse()

Returns the value of the response, which will be one of the three variables: ABORT, IGNORE, or RETRY.

retry

public final boolean isRetry()

Tests to see if the response is RETRY. If it returns **true**, the response is RETRY.

ErrorResponse methods

Method	Implemented in
abort()	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
ignore()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
retry()	this class
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

abort()

public final void abort()

Returns a *response* value of ABORT. Call this method to cause the operation to fail with an exception throw.

ignore()

public final void ignore()

Returns a *response* value of IGNORE. Call this method to cause the operation to fail without an exception throw.

retry()

public final void retry()

Returns a *response* of RETRY. Call this method to attempt to retry the operation that failed.

EventMulticaster class

jb.util package

Extends java.lang.Object

EventMulticaster is a multicaster for all user-defined events. All events, such as model and selection events, use *EventMulticaster*, which sends an event to all listeners. *EventMulticaster* maintains an array of listeners. The *add()*, *remove()*, and *find()* methods maintain this list.

The *hasListeners()* method determines whether any objects are listening for events. *EventMulticaster* has a *dispatch()* method and two specialized dispatch methods: *exceptionDispatch()*, which is used for events that can throw exceptions, and *vetoableDispatch()*, which is used for events that can decline an event.

EventMulticaster variables

Variable	Defined in
listeners	this class

listeners

protected transient EventListener[] listeners

The array of action listeners.

EventMulticaster properties

Property	Implemented in
class*	java.lang.Object
listenerCount*	this class

listenerCount

public int getListenerCount()

Returns the number of event listeners in the array of listeners.

EventMulticaster methods

Method	Implemented in
add(com.borland.jb.util.EventMulticaster, java.util.EventListener)	this class
add(java.util.EventListener)	this class
clone()	java.lang.Object
dispatch(com.borland.jb.util.DispatchableEvent)	this class
equals(java.lang.Object)	java.lang.Object
exceptionDispatch(com.borland.jb.util.ExceptionDispatch)	this class
finalize()	java.lang.Object
find(java.util.EventListener)	this class
hashCode()	java.lang.Object
hasListeners()	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
remove(com.borland.jb.util.EventMulticaster, java.util.EventListener)	this class
remove(java.util.EventListener)	this class
toString()	java.lang.Object
vetoableDispatch(com.borland.jb.util.VetoableDispatch)	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

add(com.borland.jb.util.EventMulticaster, java.util.EventListener)

```
public static final EventMulticaster add(EventMulticaster caster, EventListener listener)
```

Adds an object to the array of listeners. It is used for efficiency to avoid allocating an *EventMulticaster* until there are listeners. The allocated *EventMulticaster* can be tested for **null** to see if there are any listeners, thereby improving efficiency again.

caster The multicaster that is allocated.

listener The object that is added to the list of listeners for events.

add(java.util.EventListener)

```
public final synchronized void add(EventListener listener)
```

Adds an object to the array of listeners.

listener The object that is added to the list of listeners for events.

dispatch(com.borland.jb.util.DispatchableEvent)

```
public final void dispatch(DispatchableEvent e)
```

Sends a *DispatchableEvent* to all listeners. This method is a high-speed dispatcher that does not need to be synchronized.

e The dispatchable event sent to all listeners.

exceptionDispatch(com.borland.jb.util.ExceptionDispatch)

```
public final void exceptionDispatch(ExceptionDispatch e)
```

Sends an *ExceptionDispatch* event to all listeners. It is used for all events that can throw exceptions. This method is a high-speed dispatcher that does not need to be synchronized.

e The exception event sent to all listeners.

find(java.util.EventListener)

```
public int find(EventListener listener)
```

Searches for the specified listener among the array of listening objects.

listener The object being searched for in the list of listeners.

hasListeners()

```
public final boolean hasListeners()
```

Determines if there are any listeners for events. If the method returns **true**, one or more event listeners are present.

remove(com.borland.jb.util.EventMulticaster, java.util.EventListener)

```
public static final EventMulticaster remove(EventMulticaster caster, EventListener listener)
```

Removes the specified listening object from the array of event listeners. This *remove()* method is the counterpart of the *add()* method that allocates an event multicaster and improves efficiency.

caster The event multicaster object.

listener The listening object that is removed from the array of event listeners.

remove(java.util.EventListener)

```
public final synchronized void remove(EventListener listener)
```

Removes the specified listening object from the array of event listeners.

listener The listening object that is removed from the array of event listeners.

vetoableDispatch(com.borland.jb.util.VetoableDispatch)

```
public final boolean vetoableDispatch(VetoableDispatch e)
```

Sends a *VetoableDispatch* event to all listeners. It is used for all events that can throw *VetoException* and therefore decline the event. This method is a high-speed dispatcher that does not need to be synchronized.

vetoableDispatch() returns **false** if an event listener throws a *VetoException*. A return value of **true** indicates the listener accepted the event.

e The *VetoableDispatch* event sent to all listeners.

ExceptionChain component

jb.util package

Extends java.lang.Object

Implements java.io.Serializable

The *ExceptionChain* component represents the node of a *ChainedException* object (the linked list of exceptions) that can be traversed.

See also *com.borland.dbswing.DBExceptionDialog*,
 com.borland.dx.dataset.DataSetException

ExceptionChain constructors

ExceptionChain()

public ExceptionChain()

Constructs a new *ExceptionChain*.

ExceptionChain properties

Property	Implemented in
class*	java.lang.Object
exception*	this class
next*	this class

exception

public Throwable getException()

Places an exception at this node.

next

public ExceptionChain getNext()

Moves to the next *ExceptionChain* instance. Returns **null** for the last instance.

ExceptionChain methods

Method	Implemented in
append(java.lang.Throwable)	this class
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getOriginalMessage(java.lang.Throwable)	this class
hasExceptions()	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
printDiagnosticStackTrace()	this class
printStackTrace(java.io.PrintStream)	this class
toString()	java.lang.Object
wait()	java.lang.Object

Method	Implemented in
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

append(java.lang.Throwable)

public void append(Throwable ex)

Appends the given exception to the end of the *ChainedException* object (the linked list of exceptions).

getOriginalMessage(java.lang.Throwable)

public static String getOriginalMessage(Throwable ex)

Gets the original exception in the *ChainedException* object (the linked list of exceptions).

hasExceptions()

public boolean hasExceptions()

Returns **true** if there is an exception instance in the chain.

printDiagnosticStackTrace()

public void printDiagnosticStackTrace()

Prints all diagnostic stack traces in the chain.

printStackTrace(java.io.PrintStream)

public void printStackTrace(java.io.PrintStream out)

Prints all exception stack traces in the chain.

out The chain to print.

ExceptionDispatch interface

jb.util package

Implemented by com.borland.dx.dataset.DataChangeEvent,
 com.borland.dx.dataset.MasterUpdateEvent,
 com.borland.dx.sql.dataset.ConnectionUpdateEvent

The *ExceptionDispatch* interface is an interface for events that *com.borland.jb.util.EventMulticaster* can send to multiple listeners. The events can throw exceptions.

ExceptionHandlerDispatch methods

Method	Implemented in
exceptionDispatch(java.util.EventListener)	this class

exceptionDispatch(java.util.EventListener)

public void exceptionDispatch(EventListener listener)

Sends an event that to the specified listener. The event throws an exception. The exception object is *Exception*.

listener The object listening for the event.

ExceptionHandler interface

jb.util package

The *ExceptionHandler* interface allows an object to generically handle exceptions.

ExceptionHandler methods

Method	Implemented in
handleException(java.lang.Exception)	this class

handleException(java.lang.Exception)

public void handleException(Exception ex)

This method processes the exception as appropriate.

x The exception to handle.

FastStringBuffer component

jb.util package

Extends java.lang.Object

Use the *FastStringBuffer* component to replace use of the *StringBuffer* class in those instances when a buffer is not shared. It removes some of the complications of synchronization and sharing.

Warning Because none of the *FastStringBuffer* methods are synchronized, these methods should not be used for objects which may be accessed simultaneously. *FastStringBuffer* is intended for rapid processing on local objects.

FastStringBuffer variables

Variable	Defined in
NOT_A_CHAR	this class
NOTACHAR	this class

NOT_A_CHAR

public static final int NOT_A_CHAR = 0

The given character is invalid.

NOTACHAR

public static final int NOTACHAR = 0

A fetch has run out of bounds.

FastStringBuffer constructors

FastStringBuffer()

public FastStringBuffer()

Constructs a *FastStringBuffer* object with a default length of 16 characters (char[16]). The offset and count are initialized to 0.

FastStringBuffer(char, int)

public FastStringBuffer(char c, int nChars)

Constructs *FastStringBuffer* using the given number of repetitions and the given character. The initial capacity of the buffer is equal to the number of repetitions.

c The character to construct *FastStringBuffer* with.

nChars The number of repetitions.

FastStringBuffer(char[])

```
public FastStringBuffer(char[] cArray)
```

Constructs *FastStringBuffer* with the given character. The initial capacity of the buffer is equal to the given character plus 16.

cArray The character to construct *FastStringBuffer* with.

FastStringBuffer(char[], int, int)

```
public FastStringBuffer(char[] cArray, int offset, int len)
```

Constructs *FastStringBuffer* from the given character with the specified number of characters. The initial capacity of the buffer is equal to the specified number of characters plus 16.

cArray The character to construct *FastStringBuffer* with.

offset The location of *FastStringBuffer*.

len The length of the buffer.

FastStringBuffer(int)

```
public FastStringBuffer(int length)
```

Constructs an empty *FastStringBuffer* of the given length.

length The initial capacity.

FastStringBuffer(java.lang.String)

```
public FastStringBuffer(String str)
```

Constructs a new *FastStringBuffer* from the given string. The initial contents of the buffer are a copy of *str*. The initial capacity of the buffer is equal to the length of the string plus 16.

str The string to create *FastStringBuffer* from.

FastStringBuffer properties

Property	Implemented in
class*	java.lang.Object
length	this class
offset	this class
value*	this class

length

```
public int getLength()
public void setLength(int newLength)
```

Number of characters in the *FastStringBuffer*.

offset

```
public int getOffset()
public void setOffset(int offset)
```

Current position in *FastStringBuffer*. See *nextChar()* for more information.

value

```
public char[] getValue()
```

Returns the *char[]* storage used by *FastStringBuffer*.

FastStringBuffer methods

Method	Implemented in
append(char, int)	this class
append(char)	this class
append(char[], int, int)	this class
append(char[])	this class
append(com.borland.jb.util.FastStringBuffer)	this class
append(java.lang.Object)	this class
append(java.lang.String)	this class
capacity()	this class
charAt(int)	this class
charFromString(java.lang.String)	this class
charToUnicodeEscape(char)	this class
clone()	java.lang.Object
currentChar()	this class
empty()	this class
equals(java.lang.Object)	java.lang.Object
expandDelimiters(java.lang.String, java.lang.String)	this class
finalize()	java.lang.Object
firstChar()	this class
getChars(int, int, char[], int)	this class
hashCode()	java.lang.Object
indexOf(com.borland.jb.util.FastStringBuffer, int)	this class
IndexOfSubString(com.borland.jb.util.FastStringBuffer, int)	this class

Method	Implemented in
insert(int, boolean)	this class
insert(int, char)	this class
insert(int, char[])	this class
insert(int, java.lang.Object)	this class
insert(int, java.lang.String)	this class
lastChar()	this class
lastIndexOf(com.borland.jb.util.FastStringBuffer, int)	this class
length()	this class
makeroom(int)	this class
nextChar()	this class
normalizeDelimiters(java.lang.String)	this class
notify()	java.lang.Object
notifyAll()	java.lang.Object
offset()	this class
parseBackSlash()	this class
parseLiteral()	this class
parseLiteral(char, boolean)	this class
peekNextChar()	this class
priorChar()	this class
removeChar()	this class
removeCharAt(int)	this class
removeChars(int)	this class
removeCharsAt(int, int)	this class
replaceCharAt(int, char)	this class
setCharAt(int, char)	this class
sourceToText(java.lang.String)	this class
stringFromChar(char)	this class
substring(int, int)	this class
textToSource(java.lang.String, boolean, java.lang.String)	this class
textToSource(java.lang.String, boolean)	this class
toString()	this class
value()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

append(char)

```
public FastStringBuffer append(char c)
```

Appends the string representation of the specified character to *FastStringBuffer*. The length of the buffer is increased by 1.

c The character to append.

append(char, int)

```
public FastStringBuffer append(char c, int appendCount)
```

Appends the given number of repetitions of the given character to *FastStringBuffer*. The length of the buffer is increased by *appendCount*.

c The character to append.

appendCount The number of times to append the character.

append(char[])

```
public FastStringBuffer append(char[] str)
```

Appends the string representation of the specified string to *FastStringBuffer*. The length of the buffer is increased by the number of characters in the string.

str The string to append.

append(char[], int, int)

```
public FastStringBuffer append(char[] str, int offset, int len)
```

Appends the string representation of the specified string at the given offset for the given count. *FastStringBuffer* is increased by the length of the string.

str The string to append.

offset The offset at which to append the string.

len The length of the string.

append(com.borland.jb.util.FastStringBuffer)

```
public FastStringBuffer append(FastStringBuffer fsb)
```

Appends one *FastStringBuffer* to another one. The original *FastStringBuffer* is increased by the length of the new buffer.

fsb The buffer to append.

append(java.lang.Object)

```
public FastStringBuffer append(Object obj)
```

Appends the given object to *FastStringBuffer*.

obj The object to append.

append(java.lang.String)

```
public FastStringBuffer append(String str)
```

Appends the string representation of the specified string to *FastStringBuffer*.

str The string to append.

capacity()

```
public int capacity()
```

Returns the current capacity of *FastStringBuffer*.

charAt(int)

```
public char charAt(int index)
```

Returns the character at the given position.

index The index position to examine. Must be greater than or equal to 0, and less than the length of *FastStringBuffer*.

charFromString(java.lang.String)

```
public static char charFromString(String s)
```

Returns the first “logical” char value from the given *String*. This means that it handles backslashes, Unicode escape sequences, and so on.

charToUnicodeEscape(char)

```
public static String charToUnicodeEscape(char ch)
```

Returns a *String* containing a Unicode escape sequence representing the given character. For example, `charToUnicodeEscape('1')` returns `new String("\\u0031")`.

ch The character to be converted to a Unicode escape sequence.

currentChar()

```
public char currentChar()
```

Returns the character at *offset*. Note that this method starts at the current *offset*.

This method returns *NOTACHAR* if empty.

empty()

```
public void empty()
```

Nulls the *FastStringBuffer*.

expandDelimiters(java.lang.String, java.lang.String)

```
public static FastStringBuffer expandDelimiters(String sourceString, String delimiters)
```

Creates a copy of the given *FastStringBuffer*, but translates any characters in the specified delimiter set into Unicode “escape” sequences. This allows the new *FastStringBuffer* to use the normal *StringTokenizer* for parsing. This method returns a new *FastStringBuffer* containing all the characters of the source *String*, but with all delimiters expanded to Unicode escape sequences.

<i>sourceString</i>	The <i>String</i> to be scanned and converted (this <i>String</i> itself is not altered).
<i>delimiters</i>	<p>A <i>String</i> consisting of the delimiters you don’t want to see in the output <i>StringBuffer</i>, for example,</p> <pre>new String("\t\r\n,")</pre> <p>Wherever these occur in the <i>sourceString</i>, they are converted to a Unicode escape sequence.</p>

firstChar()

```
public char firstChar()
```

Sets *offset()* to 0 and returns to the first character in the buffer. This method is used for loops with *lastChar()*.

getChars(int, int, char[], int)

```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

Copies characters from *FastStringBuffer* into the destination character array.

The first character to be copied is at *srcBegin*. The last character to be copied is at *srcEnd*-1; making the total number of characters to be copied *srcEnd* - *srcBegin*. The characters are copied into the subarray of *dst* starting at *dstBegin* and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

This method throws a *StringIndexOutOfBoundsException* if:

- *srcBegin* is negative.
- *srcBegin* is greater than *srcEnd*.
- *srcEnd* is greater than the length of this string.
- *dstBegin* is negative.
- *dstBegin* + (*srcEnd* - *srcBegin*) is larger than *dst.length*.

srcBegin The index of the first character in the string to copy.
srcEnd The index after the last character in the string to copy.
dst The destination array.
dstBegin The start offset in the destination array.

indexOf(com.borland.jb.util.FastStringBuffer, int)

```
public int indexOf(FastStringBuffer subStr, int fromIndex)
```

Returns the index within *FastStringBuffer* of the first occurrence of *subStr*, starting at the specified index.

There is no restriction on the value of *fromIndex*:

- If it is negative, the entire string is searched.
- If it is 0, the entire string is searched.
- If it is greater than the length of the string, -1 is returned.
- If it is equal to the length of this string: -1 is returned.

subString The substring to search for.
fromIndex The index to start the search from.

IndexOfSubString(com.borland.jb.util.FastStringBuffer, int)

```
public int IndexOfSubString(FastStringBuffer subStr, int fromIndex)
```

Finds the index in the given *FastStringBuffer* where the specified sub-string begins. Returns -1 if the string is not found.

subStr The string to look for.
fromIndex The index to start the search from.

insert(int, boolean)

```
public FastStringBuffer insert(int offset, boolean b)
```

Inserts the string representation of the given boolean into *FastStringBuffer* at the given offset position.

The length of the buffer is increased by the length of *b*.

offset The position at which to insert the boolean.
b The boolean value to insert.

insert(int, char)

```
public FastStringBuffer insert(int offset, char c)
```

Inserts the string representation of the given character into *FastStringBuffer* at the given position.

The length of the buffer is increased by 1.

<i>offset</i>	The position at which to insert the character.
<i>c</i>	The character to insert.

insert(int, char[])

```
public FastStringBuffer insert(int offset, char[] str)
```

Inserts the string representation of the specified character string into *FastStringBuffer* at the given offset position.

The length of the buffer increases by the length of the string.

<i>offset</i>	The position at which to insert the character string.
<i>str</i>	The character string to insert.

insert(int, java.lang.Object)

```
public FastStringBuffer insert(int offset, Object obj)
```

Inserts the string representation of the given object into *FastStringBuffer* at the given offset position.

The length of the buffer increases by the length of the object.

<i>offset</i>	The position at which to insert the object.
<i>obj</i>	The object to insert.

insert(int, java.lang.String)

```
public FastStringBuffer insert(int offset, String str)
```

Inserts the string representation of the given string into *FastStringBuffer* at the given offset position.

The length of the buffer increases by the length of the string.

<i>offset</i>	The position at which to insert the string.
<i>str</i>	The string to insert.

lastChar()

public char lastChar()

Moves the *offset()* to the last character in *FastStringBuffer* and returns that character. Meant to be used as *lastChar()/priorChar()* loop. This method returns *NOTACHAR* if empty.

lastIndexOf(com.borland.jb.util.FastStringBuffer, int)

public int lastIndexOf(FastStringBuffer subStr, int fromIndex)

Returns the index within *FastStringBuffer* of the last occurrence of *subStr*. The returned index indicates the start of the substring, and it must be equal to or less than *fromIndex*.

There is no restriction on the value of *fromIndex*:

- If it is negative, -1 is returned.
- If it is -1, -1 is returned.
- If it is greater than the length of the string, the entire string is searched.
- If it is equal to the length of this string, the entire string is searched.

subStr

The string to look for.

fromIndex

The index to start the search from.

length()

public int length()

Returns the number of actual characters in *FastStringBuffer*.

makeroom(int)

public void makeroom(int minimumCapacity)

Increases the size of the *FastStringBuffer* so that it will hold at least the given number of characters.

minimumCapacity

The minimum number of characters in the buffer.

nextChar()

public char nextChar()

Moves the *offset()* to the next sequential character in *FastStringBuffer* and returns that character.

normalizeDelimiters(java.lang.String)

```
public FastStringBuffer normalizeDelimiters(String delimiters)
```

Turns any Unicode escape sequences that would result in one of the given delimiter characters into the displayable form of that delimiter. This method is the opposite of *expandDelimiters*.

Note Do NOT pass any non-displayable delimiters into this method, for example, `'\r'`.

delimiters

The Unicode escape sequences.

offset()

```
public int offset()
```

Internal current position in *FastStringBuffer* used by the following methods:

- *firstChar()*
- *lastChar()*
- *nextChar()*
- *priorChar()*

parseBackSlash()

```
public char parseBackSlash()
```

Given a *FastStringBuffer* where *charAt()*, *nextChar()* or *priorChar()* have just returned the backslash character (in other words, where *value[offset] == '\\'*), this routine parses the rest as a single character backslash value (for example, `"?"`) and advances the offset. It returns that character and leaves the *FastStringBuffer* pointing at the next character after the value.

parseLiteral()

```
public FastStringBuffer parseLiteral()
```

Given a *String* which needs to be parsed as a literal *String* (including backslash characters), and assuming that *value[offset]* is currently pointing at the starting delimiter of this *String*, this routine buffers everything up to (but not including) another delimiter like the first. It advances the offset past that delimiter so that subsequent string processing can continue. It returns a new *FastStringBuffer* containing the literal.

parseLiteral(char, boolean)

```
public FastStringBuffer parseLiteral(char delimiter, boolean allowDouble)
```

Given a string which needs to be parsed as a literal string (including backslash characters), and assuming *value[offset]* is currently pointing at the

starting delimiter of this string, this routine buffers everything up to (but not including) another delimiter like the first. It advances the offset past that delimiter so subsequent string processing can continue. It returns a new *FastStringBuffer* containing the literal.

delimiter The char value which marks the end of the literal.

allowDouble A value of **true** indicates that two delimiters specified in a row evaluate to a single occurrence of that literal in the string (and that it is not a delimiter).

peekNextChar()

public char peekNextChar()

Peeks at the next character *without* advancing any pointers. Used in a *firstChar()/nextChar()* type loop.

priorChar()

public char priorChar()

Moves the *offset()* to the previous sequential character in *FastStringBuffer* and returns that character. Meant to be used as *lastChar()/priorChar()* loop. This method returns *NOTACHAR* if empty.

removeChar()

public void removeChar()

Removes the “current” character from the buffer, where “current” is defined by ‘offset’. It is intended to be used in a *firstChar()/nextChar()* loop. It adjusts ‘offset’ so that the next *nextChar()* method call functions properly.

removeCharAt(int)

public void removeCharAt(int index)

Removes the character from the buffer at *index*. It adjusts *index* so a *nextChar()* loop finds the character immediately following the one removed.

index The location from which to remove the character.

removeChars(int)

public void removeChars(int removeCount)

Removes the specified number of characters from the current position (where “current” is defined as *value[offset]*). It adjusts ‘offset’ so that the next *nextChar()* method call encounters the next character in the buffer.

removeCount The number of characters to remove.

removeCharsAt(int, int)

public void removeCharsAt(int index, int removeCount)

Removes the specified number of characters from the buffer at the specified buffer at *index*. It adjusts *index* so a *nextChar()* loop finds the character immediately following the one removed.

index The location from which to remove the character.

removeCount The number of characters to remove.

replaceCharAt(int, char)

public void replaceCharAt(int index, char c)

Replaces the character at the given position with the given character.

index The location at which to replace the character.

c The character to replace the current character with.

setCharAt(int, char)

public void setCharAt(int index, char ch)

Sets the character at the given index position to the given character.

index The location at which to set the character. Must be greater than or equal to 0, and less than the length of *FastStringBuffer*.

c The character to set the specified location to.

sourceToText(java.lang.String)

public static FastStringBuffer sourceToText(String source)

Translates a *String* which is compatible with source code (including leading and trailing quote, expands backslash characters, and so on) into its actual *String* representation. For example, the literal “\n” becomes the real linefeed character.

source The *String* to translate.

stringFromChar(char)

public static String stringFromChar(char c)

Returns a *String* which best represents the given character. This means that it expands it into a Unicode escape sequence if needed. This method is the opposite of *charFromString()*.

c The character to expand.

substring(int, int)

```
public FastStringBuffer substring(int startPos, int endPos)
```

Returns a new string that is a substring of this string. The substring begins at the specified *startPos* and extends to the character at *endPos* – 1. Thus the length of the substring is *startPos* – *endPos*.

startPos The beginning index, inclusive.

endPos The ending index, exclusive.

textToSource(java.lang.String, boolean)

```
public static FastStringBuffer textToSource(String text, boolean hasEscapes)
```

Converts a *String* into a form that compiles, translating special characters into their backslash-combination equivalents, and adding leading and trailing quotes.

textToSource(java.lang.String, boolean, java.lang.String)

```
public static FastStringBuffer textToSource(String text, boolean hasEscapes, String indentString)
```

Converts a *String* into a form that compiles, translating special characters into their backslash-combination equivalents, and adding leading and trailing quotes. Adds the specified number of indent spaces to the beginning of the string.

toString()

```
public String toString()
```

Converts *FastStringBuffer* to a string.

Overrides java.lang.Object.toString()

value()

```
public char[] value()
```

Returns the internal character string used by *FastStringBuffer*.

Hex class

jb.util package

Extends java.lang.Object

The *Hex* component provides static definitions of hexadecimal numbers as **char** and **byte** values.

Hex variables

Variable	Defined in
bytes	this class
chars	this class

bytes

```
public static final byte[] bytes = {
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
}
```

Hexidecimal numbers defined as **byte** values.

chars

```
public static final char[] chars = {
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
}
```

Hexidecimal numbers defined as **char** values.

Hex properties

Property	Implemented in
class*	java.lang.Object

Hex methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

LocaleUtil class

jb.util package

Extends java.lang.Object

Returns the locale identified by a single string with “locale_country_variant” as returned by *Locale.toString()*.

LocaleUtil properties

Property	Implemented in
class*	java.lang.Object

LocaleUtil methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getLocale(java.lang.String)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	java.lang.Object
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

getLocale(java.lang.String)

public static Locale getLocale(String localeString)

Returns the *locale* identified by a single string with “locale_country_variant” as returned by *Locale.toString()*. This method may ultimately be replaced by a similar service in the *java.util.Locale* package.

localeString A single string containing language, country, and variant, separated by ‘_’. A **null** or empty string yields the default *locale*. The country and variant are optional.

SearchPath class

jb.util package

Extends java.lang.Object

Encapsulates a search path (such as *classpath*), and can perform searches along that path given a relative directory and file name.

SearchPath constructors

SearchPath(java.lang.String, java.lang.String)

public SearchPath(String baseDirList, String relDir)

Constructs a *SearchPath* object, with the image’s full path.

<i>baseDirList</i>	The name of the image’s base directory.
<i>relDir</i>	The name of the image’s directory, relative to the base directory.

SearchPath properties

Property	Implemented in
class*	java.lang.Object

SearchPath methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
finalize()	java.lang.Object
getFile(java.lang.String)	this class
getPath(java.lang.String)	this class
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
toString()	this class
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object

getFile(java.lang.String)

```
public File getFile(String name)
```

Returns a *File* object containing an image's full path.

name The name of the image.

getPath(java.lang.String)

```
public String getPath(String name)
```

Returns a *String* containing an image's full path.

name The name of the image.

toString()

```
public String toString()
```

Converts this *SearchPath* object into a string representation.

Overrides java.lang.Object.toString()

TriStateProperty interface

jb.util package

The *TriStateProperty* interface is used to implement a property that can have three states: true, false, or default. The three integer variables of the interface control the property value.

TriStateProperty variables

Variable	Defined in
DEFAULT	this class
FALSE	this class
TRUE	this class

DEFAULT

```
public static final int DEFAULT = -1
```

The property value is the default value.

FALSE

public static final int FALSE = 0

The property value is **false**.

TRUE

public static final int TRUE = 1

The property value is **true**.

VetoableDispatch interface

jb.util package

Implemented by com.borland.dbswing.StatusLabelEvent

The *VetoableDispatch* interface is an interface for vetoable events that *com.borland.jb.util.EventMulticaster* can send to multiple listeners. A listener for a vetoable event can choose to not respond to the event, thereby refusing or vetoing it.

See also *com.borland.jb.util.ExceptionDispatch*

VetoableDispatch methods

Method	Implemented in
vetoableDispatch(java.util.EventListener)	this class

vetoableDispatch(java.util.EventListener)

public void vetoableDispatch(EventListener listener)

Sends an event to the specified listener. The listener can choose to ignore the event and not respond to it by throwing *VetoException*.

listener The object listening for the event.

VetoException component

jb.util package

Extends java.lang.Exception

Implements java.io.Serializable

A *VetoException* object is thrown when an *EventListener* wishes to halt the multicasting of a vetoable event.

See also *com.borland.jb.util.EventMulticaster*, *com.borland.jb.util.VetoableDispatch*

VetoException constructors

VetoException()

public VetoException()

Constructs an *VetoException* object with no detail message.

VetoException(java.lang.String)

public VetoException(String vetoMessage)

Provides the application the opportunity to pass a message for default error message handling. Many event processors will ignore the *vetoMessage*.

vetoMessage A string containing the message for default error handling. A *vetoMessage* is not required.

VetoException properties

Property	Implemented in
class*	java.lang.Object
localizedMessage*	java.lang.Throwable
message*	java.lang.Throwable
vetoMessage*	this class

vetoMessage

public String getVetoMessage()

Returns the message used for default error handling.

VetoException methods

Method	Implemented in
clone()	java.lang.Object
equals(java.lang.Object)	java.lang.Object
fillInStackTrace()	java.lang.Throwable
finalize()	java.lang.Object
hashCode()	java.lang.Object
notify()	java.lang.Object
notifyAll()	java.lang.Object
printStackTrace()	java.lang.Throwable
printStackTrace(java.io.PrintStream)	java.lang.Throwable
printStackTrace(java.io.PrintWriter)	java.lang.Throwable
toString()	java.lang.Throwable
wait()	java.lang.Object
wait(long, int)	java.lang.Object
wait(long)	java.lang.Object



Data type conversions

Data type conversions during resolving

During the resolving phase when you save changes made to the data back to its source, the data is converted from *Variant* data types to JDBC types as follows:

- 1 From the conversion from the user-specified data type to the default Variant data type
- 2 From mapping of Variant data types to JDBC data types, if necessary

Conversion from the user-specified data type to the default Variant data type

If you set a column's *dataType* property, thereby over-riding the default JDBC-to-*Variant* mapping, automatic data type coercion is done, for example, in the case of persistent columns. The exception to the automatic type coercion is when converting to and from a *String* to any type other than *String* (and other conversions involving such differing data types). A *VariantException* is thrown in these cases.

To customize the coercion, or to prevent the *VariantException* from being thrown (in cases as described in the previous paragraph), wire the *CoerceFromListener.coerceFromColumn()* event.

When you write an event handler for *CoerceFromListener*, you typically need to write one for *CoerceToListener* as well. If your *DataSet* is editable and uses the default *UpdateMode* option, *JDataStore* will generate update queries that look for rows on the server that match the original data of the rows that were edited. Without coercion to convert field values back to what they looked like before the *CoerceToListener* modified them, these update queries will

likely fail. Note that this will happen even if you don't edit values in the column with the *CoerceToListener* because update queries, by default, compare all searchable columns in rows on the server.

See also *Data type conversions during data providing, CoerceToListener.coerceToColumn()* event

Mapping of Variant data types to JDBC data types

When resolving changes made to the local copy of data back to its *Database* source, *Variant* data types are translated into corresponding JDBC data types according to the table below. The JDBC data types listed are defined in *java.sql.Types*. The *Variant* data types are defined in *com.borland.dx.dataset.Variant*.

Any other value specified for a *Variant* data type generates a *DataSetException* of *UnrecognizedDataType*.

Table A.1 Variant data type translation

Variant data type	Conversion process to JDBC type
<i>Variant.BIGDECIMAL</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setBigDecimal()</i> method.
<i>Variant.BINARY</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setBinaryStream()</i> method.
<i>Variant.BOOLEAN</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setBoolean()</i> method.
<i>Variant.DATE</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setDate()</i> method.
<i>Variant.DOUBLE</i>	If the column's <i>sqlType</i> is <i>java.sql.Types.REAL</i> or <i>java.sql.Types.FLOAT</i> , the value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setFloat()</i> method. Otherwise, the value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setDouble()</i> method.
<i>Variant.FLOAT</i>	If the column's <i>sqlType</i> is <i>java.sql.Types.REAL</i> or <i>java.sql.Types.FLOAT</i> , the value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setFloat()</i> method. Otherwise, the value will be bound using the <i>java.sql.PreparedStatement</i> object's <i>setDouble()</i> method.
<i>Variant.INT</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setInt()</i> method.
<i>Variant.LONG</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setLong()</i> method.
<i>Variant.OBJECT</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setObject()</i> method.

Table A.1 Variant data type translation (continued)

Variant data type	Conversion process to JDBC type
<i>Variant.STRING</i>	<p>If the Column's <i>sqlType</i> is <i>java.sql.Types.LONGVARCHAR</i>, the value is bound as <i>java.sql.Types.LONGVARCHAR</i>.</p> <p>If the column's <i>sqlType</i> is <i>java.sql.Types.VARCHAR</i>, the value is bound as <i>java.sql.Types.VARCHAR</i>.</p> <p>If the column has precision (in other words, precision does not equal -1), the string is right-padded with spaces up to length specified by precision and is bound as <i>java.sql.Types.CHAR</i>. For example, if the value is "cat" and the column's precision is 5, the value will be bound as "cat ".</p> <p>Otherwise, the value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setString()</i> method.</p>
<i>Variant.TIMESTAMP</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setTimestamp()</i> method.
<i>Variant.TIME</i>	The value is bound using the <i>java.sql.PreparedStatement</i> object's <i>setTime()</i> method.

Data type coercions

In addition to the initial *Variant* to JDBC data type conversion, a second data type translation phase occurs where the JDBC-typed data is saved back to the data source.

If a column's *dataType* is not the same data type as that of the data source, automatic data type coercion occurs. The exception to the automatic type coercion is when converting to and from a *String* to any type other than *String* (and other conversions involving such differing data types). A *VariantException* is thrown in these cases.

To customize the coercion, or to prevent the *VariantException* from being thrown in cases as noted in the previous paragraph, wire the *CoerceFromListener.coerceFromColumn()* event.

See also *Data type conversions during data providing*, *CoerceToListener.coerceToColumn()* event

Data type conversions during data providing

During the providing phase, data is converted from JDBC data types to *Variant* data types. This involves the following:

- 1 Mapping of JDBC data types to Variant data types
- 2 Data type coercions occur, if necessary

Mapping of JDBC data types to Variant data types

When you obtain data from the data source, the JDBC data types are translated into corresponding *Variant* data types. The JDBC data types are listed below and are defined in *java.sql.Types*. The *Variant* data types are defined in *com.borland.dx.dataset.Variant*.

Precision, scale and type have default values if they are not explicitly set. Precision and scale default to -1. Type defaults to *Variant.STRING*. The *sqlType* is based on the value returned from the JDBC driver.

For information on data mapping of SQL server physical types to JBDC data types, check your driver documentation.

Table A.2 JDBC data type translation

JDBC type	Maps to Variant data type
<i>java.sql.Types.BIGINT</i>	<i>Variant.LONG</i>
<i>java.sql.Types.BINARY</i>	<i>Variant.BINARY</i> with precision
<i>java.sql.Types.BIT</i>	<i>Variant.BOOLEAN</i>
<i>java.sql.Types.CHAR</i>	<i>Variant.STRING</i> with precision
<i>java.sql.Types.DATE</i>	<i>Variant.DATE</i>
<i>java.sql.Types.DECIMAL</i>	<i>Variant.BIGDECIMAL</i> with scale and precision
<i>java.sql.Types.DOUBLE</i>	<i>Variant.DOUBLE</i>
<i>java.sql.Types.FLOAT</i>	<i>Variant.DOUBLE</i>
<i>java.sql.Types.INTEGER</i>	<i>Variant.INT</i>
<i>java.sql.Types.LONGVARBINARY</i>	<i>Variant.BINARY</i> without precision
<i>java.sql.Types.LONGVARCHAR</i>	<i>Variant.STRING</i> without precision (BLOB columns typically have an open-ended length)
<i>java.sql.Types.NUMERIC</i>	<i>Variant.BIGDECIMAL</i> with scale and precision
<i>java.sql.Types.OTHER</i>	<i>Variant.OBJECT</i>
<i>java.sql.Types.REAL</i>	<i>Variant.DOUBLE</i>
<i>java.sql.Types.SMALLINT</i>	<i>Variant.SHORT</i>
<i>java.sql.Types.TIME</i>	<i>Variant.TIME</i>
<i>java.sql.Types.TIMESTAMP</i>	<i>Variant.TIMESTAMP</i>
<i>java.sql.Types.TINYINT</i>	<i>Variant.BYTE</i>
<i>java.sql.Types.VARBINARY</i>	<i>Variant.BINARY</i> without precision
<i>java.sql.Types.VARCHAR</i>	<i>Variant.STRING</i> with precision

Data type coercions

In addition to the initial JDBC to *Variant* data type conversion, a second data type translation phase occurs where the Variant-typed data is stored in *Column* components.

If you set a column's *dataType* property, thereby over-riding the default JDBC-to-*Variant* mapping, automatic data type coercion is done, for

example, in the case of persistent columns. The exception to the automatic type coercion is when converting to and from a *String* to any type other than *String* (and other conversions involving such differing data types). A *VariantException* is thrown in these cases.

To customize the coercion, or to prevent the *VariantException* from being thrown, wire the *CoerceToListener.coerceToColumn()* event.

When you write an event handler for *CoerceToListener*, you typically need to write one for *CoerceFromListener* as well. If your *DataSet* is editable and uses the default *UpdateMode* option, *JDataStore* will generate update queries that look for rows on the server that match the original data of the rows that were edited. Without coercion to convert field values back to what they looked like before the *CoerceToListener* modified them, these update queries will likely fail. Note that this will happen even if you don't edit values in the column with the *CoerceToListener* because update queries, by default, compare all searchable columns in rows on the server.

See also Data type conversions during resolving,
CoerceFromListener.coerceFromColumn() event

B

String-based patterns (masks)

In DataExpress components, the pattern syntax for all edit masks, display masks, and import/export masks, and for parsing such masks, is a direct extension of that used by the format classes in the `java.text` package. There are separate patterns and symbols for Numeric data patterns, Date/time data patterns, Text data patterns and boolean data (see `com.borland.dx.text.BooleanFormat`).

Some of these symbols, such as the braces to indicate optional portions, are additions to the JDK. The DataExpress classes ensure that such extensions are never sent to the JDK classes (which would not understand them). The extensions are used and then stripped from the pattern.

Note For ease of use, the symbol definitions are designed to be compatible with those in Borland's Delphi product. In those cases where Visual Basic is different from, but not incompatible with Delphi, those symbols are defined as well.

Numeric data patterns

A numeric pattern consists of two formats, separated by a semicolon. The second pattern, when present, determines how negative numbers are formatted and displayed.

To format the display or editing of a numeric data, use the symbols in the tables below.

Table B.1 JDK standard numeric format symbols

Symbol	Meaning	Notes
0	In an edit pattern, a required digit In a display pattern, indicates that leading zeroes are displayed.	
#	In an edit pattern, an optional digit In a display pattern, indicates that leading zeroes do not display.	
. (period)	Placeholder for the decimal separator.	This is not necessarily the actual character displayed as the separator. That is determined by your Locale.
, (comma)	Placeholder for the grouping (thousands) separator, showing the grouping to be used.	This is not necessarily the actual character displayed as the separator. That is determined by your Locale.
; (semicolon)	Format separator. Place a semicolon between the format pattern for a positive number (first) and the pattern for a negative number (second).	
%	Divide entry by 100 and show as a percentage.	
- (hyphen)	A leading minus sign appears for negative numbers.	Do not use together with parentheses.
()	The entire expression is enclosed in parentheses to show that it is negative, for example: (\$#,###.##).	Do not use together with the hyphen.
\nnnn	Single character literal.	Enter the octal, hexadecimal, or Unicode number for the desired character. This symbol escapes even out of single quotes.

Table B.2 DataExpress additional numeric format symbols

Symbol	Meaning	Notes
{ }	Use to bracket optional portions of the pattern.	Entering #####.{##} allows the user to choose whether to enter the decimal fraction.
^ (carat)	Sets the initial cursor position when editing.	
' (single quote)	Enclose text constants. This method of including constants is preferred to the other symbol styles, as being more reliable.	'Lira' yields the words "Lira" in the data.

Date/time data patterns

To format the display or editing of a date or timestamp data, use the symbols in the tables that follow.

Table B.3 JDK standard date/time format symbols

Symbol	Meaning	Presentation
G	Era designator.	A “G” in the pattern yields AD.
y	Year. Enter four “y”s to format to all four digits of the year. Enter “y” or “yy” to yield the last two digits of the year.	yyyy yields 1996 yy yields 96.
M	The month of the year.	“M” yields the number of the month (February would be 2). “MM” yields the number of the month, but single digits have leading zeroes (02). “MMM” yields the abbreviation of the month (Feb). “MMMM” yields the full name of the month (February).
d	The day of the month.	For Christmas Day, entering: “d” yields 25 “dd” yields 25 “ddd” yields 025 “dddd” yields 0025.
h	The hour in the day, using a 12-hour clock.	1 through 12
H	The hour in the day, using a 24-hour clock.	0 through 23
m	The minute in the hour.	1 through 59
s	The second in the minute.	1 through 59
S	The millisecond.	1 through 999
E	The day in the week.	“EEEE” yields the full name of the day, e.g., Tuesday. “EEE” (or fewer) yields the abbreviation, e.g., Tues.
D	The day in the year, as a Julian date (number).	July 7 would be 189
F	The ordinal of the day of the week in the month. For example, the second Tuesday in July.	For January 7, 1997, entering: “F” yields 1 “FF” yields 01 “FFF” yields 001 “FFFF” yields 0001.
w	The week in the year, as a number.	
W	The week in the month.	
a	Marker for am/pm.	For 1600 hours, displays 8:00 pm.
k	The hour in the day, using a 24-hour clock.	1 through 24

Table B.3 JDK standard date/time format symbols (continued)

Symbol	Meaning	Presentation
K	The hour in the day, using a 12 hour clock.	0 through 11
z	Time zone	Entering one through three “z”s yields the abbreviation (PST) Entering four “z”s yields the completely spelled out time zone (Pacific Standard Time).
' (single quote)	Enclose text constants. This method of including constants is preferred of the other symbol styles, as being more reliable.	'Area Code' yields the words “Area Code” in the data.
“ (two single quotes)	Indicates a literal single quote as a part of a text constant.	
\nnnn	Single character literal.	Enter the octal, hexadecimal, or Unicode number for the desired character. This symbol escapes even out of single quotes.

Table B.4 DataExpress additional date/time format symbols

Symbol	Meaning	Presentation
{}	Use to bracket optional portions of the pattern.	
^ (carat)	Sets the initial cursor position when editing.	

Text data patterns

Patterns for formatting and editing text data are specific to DataExpress classes. They consist of up to four parts, separated by semicolons. Only the first part is required. To include an optional third part, but not the second or fourth, enter the semicolons for each part. The parts include

- 1 The actual edit or display mask.
- 2 A zero or one (0 or 1), indicating whether literals should be stripped before posting information to a database. Zero indicates the literals should be stripped. If this part is omitted, literals are not stripped.
- 3 The character to use as a “blank” indicator. This character indicates the spaces to be filled in the data. If this part is omitted, the underscore character is used.
- 4 The character to be used to replace blank positions on output. If this part of the mask is omitted, blank positions are stripped.

To format the display or editing of a text data, use the symbols in the following table for the first part of the pattern.

Table B.5 DataExpress text data format symbols

Symbol	Meaning	Presentation
0	A digit. 0 – 9, entry required. Plus (+) and minus (–) signs not allowed.	
9	A digit. 0 – 9, entry optional. A space, plus (+) and minus (–) signs are not allowed.	
#	A digit or space, entry optional. Blank positions are converted to spaces. Plus (+) and minus (–) signs allowed.	
L	A letter. A through Z, entry required.	
l	A letter. A through Z, entry optional.	
?	A letter. A through Z, entry optional.	
A	A letter or digit, entry required.	
a	A letter or digit, entry optional.	
C	A character or space, entry required.	
c	A character or space, entry optional.	
&	A character or space, entry required.	
<	Converts the characters that follow to lower case.	
>	Converts the characters that follow to upper case.	
!	For an edit mask, makes the data fill from right to left, rather than from left to right, when characters to the left are optional. Can be placed anywhere in the edit mask.	
\	Causes the character immediately following the back slash to be displayed as a literal. Use to display any of the syntax symbol characters.	
\nnnn	Single character literal.	Enter the octal, hexadecimal, or Unicode number for the desired character. This symbol escapes even out of single quotes.
' (single quote)	Enclose character constants. For example, "990' units sold'" should display as "27 units sold".	
*	Enclose a password encrypted string. For example, "*AAAAaaa*" describes a password of at least four, and at most eight alphanumeric characters. They echo as "*" as they are entered.	
{ }	Enclose optional portions of a pattern.	
^ (carat)	Sets the initial cursor position when editing.	

Index

Numerics

3-tier applications 4-114

A

ABORT

ErrorResponse component 8-25

abort

ErrorResponse component 8-26, 8-27

access

DataSet class 4-113

DataSetView component 4-151

ProcedureDataSet class 5-44

QueryDataSet class 5-70

StorageDataSet component 4-300

TableDataSet component 4-314

accessChange

AccessListener interface 4-17

AccessEvent class 4-10

AccessEvent constructor

AccessEvent class 4-12, 4-13, 4-14

accessing data 4-305

off-line users 4-305

read-only 4-205

SQL data sources 5-32

AccessListener interface 4-17

accumulateResults

ProcedureDataSet class 5-36

Provider class 4-198

QueryDataSet class 5-61

ACTIVE_STATE

DataStore component 2-9

adapter classes 5-10

CalcAggFieldsAdapter 4-23

ColumnChangeAdapter 4-54

ColumnPaintAdapter 4-59

ConnectionUpdateAdapter 5-10

DataChangeAdapter 4-72

EditAdapter 4-152

OpenAdapter 4-183

ResolverAdapter 4-233

ResponseAdapter 4-241

adapter classes for datasets

overview in DataExpress library 4-6, 5-3

add

AggOperator class 4-22

CountAggOperator class 4-66

CustomAggOperator class 4-69

EventMulticaster class 8-29

RowFilterResponse class 4-252

SumAggOperator class 4-304

Variant component 4-348

addColumn

ColumnList component 4-58

ParameterRow component 4-190, 4-191

StorageDataSet component 4-290, 4-291

addDriver

Database component 5-22

addDrivers

Database component 5-23

added

EditListener interface 4-155

addError

EditListener interface 4-155

addExceptionListener

DataSetException class 4-136

adding

EditListener interface 4-156

adding columns 4-31

with metadata capacity 4-31

additionalBeanInfo

BasicBeanInfo class 8-7, 8-11

addLoadRowListener

StorageDataSet component 4-291

addRow

DataSet class 4-97

addServerStatusListener

DataStoreServer component 3-8

addTraceCategory

Diagnostic class 8-16

addUniqueColumn

StorageDataSet component 4-291

agg

Column component 4-36

AGG_OPERATOR_NOT_FOUND

DataStoreException class 2-37

AGG_STREAM

DataStore component 2-9

aggColumn

AggOperator class 4-20

aggColumnName

AggDescriptor class 4-19

aggDataSet

AggOperator class 4-20

AggDescriptor class 4-17

methods 4-19

AggDescriptor constructor

AggDescriptor class 4-18

aggOperator

AggDescriptor class 4-19

- AggOperator class 4-20
 - properties 4-21
- AGGREGATE
 - CalcType interface 4-28
- aggregate operator classes 4-4, 4-20
- aggregate operators
 - instantiating 4-20
 - overview 4-4
- aggregated columns
 - adapter for 4-23
 - listener for 4-24
- aggregations
 - customizing 4-67
 - defining type 4-17
 - on counts 4-65
 - on maximum values 4-176
 - on minimum values 4-179
 - on summary calculations 4-303
 - setting as calculation type 4-27
- aggValue
 - AggOperator class 4-21
- aliases
 - multi-table queries 5-58
- alignment
 - Column component 4-36
 - CustomPaintSite interface 4-70
- Alignment class 6-4
 - properties 6-7
- ALL
 - Load interface 5-29
 - MetaDataUpdate interface 4-178
- ALL_COLUMNS
 - UpdateMode interface 4-319
- allocateValues
 - DataSet class 4-98
- allRowIds
 - StorageDataSet component 4-278
- ALogDir
 - TxManager component 2-67
- ALREADY_LOADING
 - DataSetException class 4-119
- AMBIGUOUS_LOGDIR
 - TxException class 2-57
- ANCHOR_TOO_OLD
 - TxException class 2-57
- append
 - ExceptionChain component 8-32
 - FastStringBuffer component 8-38, 8-39
- appendException
 - DispatchableEvent class 8-23
- APPLICATION_ERROR
 - ValidationException class 4-327
- applications 5-14
 - designing 5-14
 - loading data in multi-threaded 4-165
 - populating three-tiered 4-114
- applyPattern
 - BooleanFormat component 6-17
 - TextFormat component 6-68
- arrayLength
 - Variant component 4-341
- ArrayResourceBundle class 8-3
 - variables 8-4
- arrayToProperties
 - ConnectionDescriptor class 5-9
- AS_NEEDED
 - Load interface 5-29
- asBigDecimal
 - Variant component 4-341
- asBoolean
 - Variant component 4-341
- ASCII
 - DataFileFormat class 4-79
- AsciiInputStream class 7-3
 - properties 7-3
- AsciiInputStream constructor
 - AsciiInputStream class 7-3
- AsciiOutputStream class 7-5
 - properties 7-5
- AsciiOutputStream constructor
 - AsciiOutputStream class 7-5
- asDate
 - Variant component 4-342
- asDouble
 - Variant component 4-342
- asFloat
 - Variant component 4-342
- asInt
 - Variant component 4-342
- asLong
 - Variant component 4-342
- asObject
 - Variant component 4-342
- asShort
 - Variant component 4-343
- ASSIGNED_NULL
 - Variant component 4-335
- assignedNull
 - ReadWriteRow class 4-219
 - Variant component 4-343
- AssignedNull_S
 - Variant component 4-335
- asTime
 - Variant component 4-343
- asTimestamp
 - Variant component 4-343
- asVariant
 - Variant component 4-343
- ASYNCHRONOUS
 - Load interface 5-30

- asynchronous fetching 4-163, 4-165
- asynchronous queries 5-71
 - canceling load operations 4-163
 - with stored procedures 5-45
- asynchronousExecution
 - QueryDescriptor class 5-75
- atFirst
 - DataSet class 4-98
 - RowIterator class 4-258
- atLast
 - DataSet class 4-98
 - RowIterator class 4-258
- autoCommit
 - Database component 5-17
- available
 - FastBufferedInputStream class 7-13

B

- background
 - Column component 4-36
 - CustomPaintSite interface 4-70
- BAD_PROCEDURE_PROPERTIES
 - DataSetException class 4-119
- BAD_QUERY_PROPERTIES
 - DataSetException class 4-120
- badProcedureProperties
 - DataSetException class 4-136
- badQueryProperties
 - DataSetException class 4-136
- BasicBeanInfo class 8-6
- beanClass
 - BasicBeanInfo class 8-8
- beanDescriptor
 - BasicBeanInfo class 8-12
- BIGDECIMAL
 - Variant component 4-335
- bigDecimal
 - Variant component 4-344
- BIGDECIMAL_PRECISION_ERROR
 - DataStoreException class 2-37
- BigDecimalFormatter class 6-7
- BigDecimalFormatter constructor
 - BigDecimalFormatter class 6-8
- BigDecimalType_S
 - Variant component 4-335
- BinaryFormatter component 6-10
- BinaryFormatter constructor
 - BinaryFormatter component 6-10
- binaryStream
 - Variant component 4-344
- BinaryStreamType_S
 - Variant component 4-335
- bind
 - RowIterator class 4-258, 4-259
- binding
 - changing for UI components 4-144
- binding to data sets 4-114
- BIRTHSTAMP_MISMATCH
 - TxException class 2-58
- blockSize
 - DataStore component 2-14
- BLogDir
 - TxManager component 2-67
- BOOLEAN
 - Variant component 4-335
- boolean
 - Variant component 4-344
- BooleanFormat component 6-14
- BooleanFormat constructor
 - BooleanFormat component 6-15
- BooleanFormatter component 6-19
- BooleanFormatter constructor
 - BooleanFormatter component 6-19
- BooleanType_S
 - Variant component 4-336
- BOTTOM
 - Alignment class 6-5
- buf
 - FastBufferedOutputStream class 7-15
- buildTrueFormatMask
 - VariantFormatStr class 6-77
- BYTE
 - Variant component 4-336
- byte
 - Variant component 4-344
- BYTE_ARRAY
 - Variant component 4-336
- byteArray
 - Variant component 4-344
- ByteArrayType_S
 - Variant component 4-336
- ByteFormatter class 6-21
 - properties 6-21
 - variables 6-21
- ByteFormatter constructor
 - ByteFormatter class 6-21
- bytes
 - Hex class 8-48
 - InputStreamToByteArray class 7-18
- ByteType_S
 - Variant component 4-336

C

- CALC
 - CalcType interface 4-28
- calcAggAdd
 - CalcAggFieldsListener interface 4-25
- calcAggDelete
 - CalcAggFieldsListener interface 4-26

- calcAggFields
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-70
 - StorageDataSet component 4-300
 - TableDataSet component 4-314
- CalcAggFieldsAdapter class 4-23
 - methods 4-24
 - properties 4-24
- calcAggFieldsListener
 - StorageDataSet component 4-278
- CalcAggFieldsListener interface 4-24
 - adapter for 4-23
- calcFields
 - CalcFieldsListener interface 4-27
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-70
 - StorageDataSet component 4-300
 - TableDataSet component 4-314
- calcFieldsListener
 - StorageDataSet component 4-278
- CalcFieldsListener interface 4-26
- calcType
 - Column component 4-36
- CalcType interface 4-27
- calculated columns 4-31
 - creating 4-27
 - defining aggregator behavior 4-20
 - listener for 4-26
 - locating values 4-80
- calculated fields *See* calculated columns
- calculations
 - defining aggregator behavior 4-20
 - defining type 4-27
 - performing on aggregated fields 4-23
 - sample applications 4-27
- callProcedure
 - ProcedureProvider class 5-51
 - ProcedureProvider component 5-72
- CAN_CLOSE
 - ConnectionUpdateEvent class 5-11
- canAdd
 - RowFilterResponse class 4-252
- CANCEL
 - ResponseEvent class 4-243
- cancel
 - DataSet class 4-98
 - ResponseEvent class 4-247, 4-248
 - RowIterator class 4-259
- canceled
 - EditListener interface 4-156
- canceled data loads 4-163
- cancelLoad
 - LoadCancel interface 4-163
- cancelLoading
 - DataSet class 4-98
- StorageDataSet component 4-292
- cancelOperation
 - DataSet class 4-99
 - StorageDataSet component 4-292
- canChangeConnection
 - ConnectionUpdateListener interface 5-13
- canNavigate
 - DataSet class 4-99
- CANNOT_CHANGE_COLUMN
 - DataSetException class 4-120
- CANNOT_CHANGE_COLUMN_DATA_TYPE
 - DataSetException class 4-120
- CANNOT_DITTO_EXISTING
 - ValidationException class 4-327
- CANNOT_FIND_TABLE_NAME
 - DataSetException class 4-120
- CANNOT_IMPORT_NULL_DATASET
 - DataSetException class 4-120
- CANNOT_OPEN
 - DataStoreException class 2-37
- CANNOT_ORPHAN_DETAILS
 - ValidationException class 4-328
- CANNOT_REFRESH
 - DataSetException class 4-120
- CANNOT_RESTRUCTURE
 - DataStoreException class 2-37
- CANNOT_SAVE_CHANGES
 - DataSetException class 4-120
- CANNOT_UPDATE_SCOPED_DATA_ROW
 - DataSetException class 4-120
- canSet
 - DataSet class 4-99
- CANT_CREATE_OPEN_STREAM
 - DataStoreException class 2-37
- capacity
 - FastStringBuffer component 8-39
- caption
 - Column component 4-37
- cascadeDeletes
 - MasterLinkDescriptor class 4-173
- cascadeUpdates
 - MasterLinkDescriptor class 4-173
- cascading deletes 4-170
- cascading updates 4-170
- CASE_INSENSITIVE
 - Locate interface 4-167
- CASEINSENSITIVE
 - Sort interface 4-263
- caseInsensitive
 - SortDescriptor class 4-267
 - StreamProperties class 2-48
- category
 - ResolveError class 5-94
- CENTER
 - Alignment class 6-5

- ChainedException interface 8-13
- CHANGED
 - ConnectionUpdateEvent class 5-11
- changed
 - ColumnChangeListener interface 4-56
- CHANGED_COLUMNS
 - UpdateMode interface 4-320
- changesPending
 - StorageDataSet component 4-292
- changing data 4-72
 - error handling 5-85
 - listener notification 4-55, 4-76
 - notification adapters 4-54, 4-72
 - SQL tables 5-100
- charAt
 - FastStringBuffer component 8-39
- charFromString
 - FastStringBuffer component 8-39
- chars
 - Hex class 8-48
- charToUnicodeEscape
 - FastStringBuffer component 8-39
- check
 - Diagnostic class 8-16, 8-17
- checkFrequency
 - TxManager component 2-68
- checkIfBusy
 - Provider class 4-199
 - Resolver class 4-233
- CHECKING_DATASTORE
 - StatusEvent class 4-270
- checkMasterLink
 - Provider class 4-199
 - QueryProvider component 5-79
- CLASS_NOT_FOUND_ERROR
 - DataSetException class 4-121
- classNotFoundException
 - DataSetException class 4-136
- CLEAR
 - StatusEvent class 4-271
- clearStatus
 - DataSet class 4-99
- clearValues
 - ReadWriteRow class 4-222
- clone
 - AggOperator class 4-22
 - Column component 4-51
 - Variant component 4-349
- cloneColumns
 - ColumnList component 4-59
 - StorageDataSet component 4-292
- cloneDataSetStructure
 - StorageDataSet component 4-292
- cloneDataSetView
 - DataSet class 4-99
- CLOSE
 - AccessEvent class 4-10
- close
 - AsciiInputStream class 7-4
 - AsciiOutputStream class 7-6
 - DataSet class 4-99
 - DataStoreConnection component 2-25
 - EncodedInputStream class 7-8
 - EncodedOutputStream class 7-10
 - FileOutputStream class 2-46
 - FileStream class 2-47
 - Provider class 4-200
 - QueryResolver component 5-84
 - Resolver class 4-233
 - ServerConnection class 3-11
 - SimpleCharInputStream class 7-20
 - SimpleCharOutputStream class 7-22
- closeConnection
 - Database component 5-23
- closeConnections
 - DataStoreServer component 3-9
- CLOSED
 - ConnectionUpdateEvent class 5-11
- closed
 - OpenListener interface 4-185
- closeDirectory
 - DataStoreConnection component 2-25
- closeProvider
 - StorageDataSet component 4-292
- closeStatement
 - ProcedureDataSet class 5-43
 - QueryDataSet class 5-68
- closeStatements
 - ProcedureResolver component 5-55
 - QueryResolver component 5-84
 - SQLResolver class 5-101
- closeStream
 - DataStoreConnection component 2-25
- closing
 - OpenListener interface 4-185
- closing data sets
 - adapter for JDBC objects 4-183
 - internal event for 4-10
 - listener for 4-184
- closing database connections 5-10, 5-12
- code
 - ResolveError class 5-94
 - ResponseEvent class 4-247
 - StatusEvent class 4-273
- coerceFrom
 - Column component 4-53
- coerceFromColumn
 - CoerceFromListener interface 4-29

- CoerceFromListener interface 4-28
- coerceTo
 - Column component 4-53
- coerceToColumn
 - CoerceToListener interface 4-30
- CoerceToListener interface 4-30
- coercions A-3, A-4
 - caution for 4-33
 - from source to column 4-30
- column
 - ColumnVariant class 4-64
- Column component 4-31
- column constraints 4-32
- Column constructor
 - Column component 4-34
- Column formatter
 - displayMask as 4-39
- COLUMN_ADD
 - AccessEvent class 4-11
- COLUMN_ALREADY_BOUND
 - DataSetException class 4-121
- COLUMN_CHANGE
 - AccessEvent class 4-11
- COLUMN_DROP
 - AccessEvent class 4-11
- COLUMN_MOVE
 - AccessEvent class 4-11
- COLUMN_NEEDS_RESTRUCTURE
 - DataSetException class 2-37
- COLUMN_NOT_IN_ROW
 - DataSetException class 4-121
- COLUMN_TYPE_CONFLICT
 - DataSetException class 4-121
- ColumnAware interface 4-53
- columnChange
 - Column component 4-53
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-70
 - StorageDataSet component 4-301
 - TableDataSet component 4-314
- ColumnChangeAdapter class 4-54
 - methods 4-54
 - properties 4-54
- columnChangeListener
 - Column component 4-37
- ColumnChangeListener interface 4-55
 - adapter for 4-54
- columnCount
 - ReadRow class 4-205
- columnIsVisible
 - DataSet class 4-100
- ColumnList component 4-56
- ColumnList constructor
 - ColumnList component 4-57
- columnName
 - Column component 4-37
 - ColumnAware interface 4-54
- columnPaint
 - Column component 4-53
- ColumnPaintAdapter class 4-59
 - methods 4-60
 - properties 4-59
- columnPaintListener
 - Column component 4-37
- ColumnPaintListener interface 4-60
 - adapter for 4-59
- column-related classes
 - overview in DataExpress library 4-4
- columns 4-31
 - associating with pick lists 4-193
 - binding to specific data set 4-54
 - calculated *See* calculated columns
 - caution for assigning data types 4-33
 - changing values *See* changing data
 - getting information 4-61
 - identifying in source tables 4-37
 - overriding default type mappings 4-33
 - ParameterRow component 4-186
 - ReadRow class 4-205
 - repainting 4-60
 - setting as persistent 4-32
 - setting as resolvable 4-48
 - specifying in multi-table queries 5-58
 - StorageDataSet component 4-278
 - storing data 4-80
 - StreamProperties class 2-48
 - updating values 4-60
- columnsArray
 - ColumnList component 4-58
- ColumnVariant class 4-61
 - methods 4-64
 - variables 4-61
- ColumnVariant constructor
 - ColumnVariant class 4-62
- com.borland.datastore
 - BeanInfos 2-2
- com.borland.dx.dataset
 - BeanInfos 4-2
 - internal use only classes 4-2
- com.borland.dx.sql.dataset
 - BeanInfos 5-1
 - internal use only classes 5-2
- com.borland.dx.text
 - internal use only classes 6-1
- com.borland.jb.io
 - internal use only classes 7-1, 8-1
- commit
 - Database component 5-23
 - DataSetConnection component 2-25

- COMMIT_ON_CLOSE
 - ResponseEvent class 4-243
- compareTo
 - Variant component 4-349
- comparisons 4-205
 - range 4-185
- complete
 - ConnectionDescriptor class 5-7
- component
 - ExceptionEvent class 4-161
- CONNECT
 - ServerStatus interface 3-11
- CONNECT_ERROR
 - ServerStatus interface 3-11
- connecting and providing data
 - overview in DataExpress library 5-3
- connection
 - Database component 5-17
- CONNECTION_DESCRIPTOR_NOT_SET
 - DataSetException class 4-121
- CONNECTION_NOT_CLOSED
 - DataSetException class 4-121
- connectionChanged
 - ConnectionUpdateListener interface 5-13
- connectionClosed
 - ConnectionUpdateListener interface 5-13
- ConnectionDescriptor class 5-5
- ConnectionDescriptor constructor
 - ConnectionDescriptor class 5-5, 5-6
- connectionDescriptorNotSet
 - DataSetException class 4-137
- connectionNotClosed
 - DataSetException class 4-137
- connectionOpening
 - ConnectionUpdateListener interface 5-13
- connections
 - closing 5-10, 5-12
 - debugging 5-15
 - setting properties for 5-14
 - storing SQL-related information for 5-5
 - update listener for 5-10
- connectionUpdate
 - Database component 5-27
- ConnectionUpdateAdapter class 5-9
 - methods 5-10
 - properties 5-10
- ConnectionUpdateEvent class 5-10
 - methods 5-12
 - properties 5-12
- ConnectionUpdateEvent constructor
 - ConnectionUpdateEvent class 5-11
- ConnectionUpdateListener interface 5-12
- connectionURL
 - ConnectionDescriptor class 5-7
- consistent
 - DataStore component 2-14
- constraints
 - data 4-32
- contacting Borland 1-3
- CONTAINER_DELEGATE
 - BasicBeanInfo class 8-8
- context
 - ResolveError class 5-95
- conversions A-2, A-3, A-4
- copy
 - Column component 4-51
- COPY_CASE_SENSITIVE
 - DataStoreConnection component 2-20
- COPY_IGNORE_ERRORS
 - DataStoreConnection component 2-20
- COPY_OVERWRITE
 - DataStoreConnection component 2-21
- copyStreams
 - DataStoreConnection component 2-26
- copyTo
 - ReadRow class 4-207, 4-208
- count
 - Diagnostic class 8-14
 - FastBufferedOutputStream class 7-15
- CountAggOperator class 4-65
 - properties 4-65
 - variables 4-65
- create
 - DataStore component 2-18
- createCallableStatement
 - Database component 5-23
- createFileStream
 - DataStoreConnection component 2-27
- createPreparedStatement
 - Database component 5-23
- createStatement
 - Database component 5-24
- currency
 - Column component 4-38
- currentChar
 - FastStringBuffer component 8-40
- currentUTCTimeMillis
 - Variant component 4-349
- cursorPos
 - ItemEditMaskState component 6-36
- cursors 4-276
- CustomAggOperator class 4-67
 - variables 4-68
- customizerClass
 - BasicBeanInfo class 8-8
- CustomPaintSite interface 4-69

D

- DATA

 - StreamVerifier class 2-51
- data 4-306
 - loading into datasets 5-72
 - saving 4-306
- data access
 - mobile users 4-305
- data constraints 4-32
- data files *See* data sources, files
- data filters 4-144, 4-250
- data providing
 - type conversions A-1
- data refreshes 4-60
- data retrieval *See* fetching, loading
- data rows *See* rows of data
- data sets 4-86, 4-276, 4-305
 - alternate views 4-144
 - associating pick lists with 4-193
 - binding to 4-114
 - binding to specific columns 4-54
 - closing 4-183, 4-184
 - displaying data 4-250
 - error handling 4-116
 - extracting data only 4-114
 - moving cursor 4-181, 4-183
 - opening 4-183, 4-184
 - recalculating 4-26
 - updating data 4-72
- data sources 4-86
 - for off-line users 4-305
 - read/write behavior 4-77
 - resolving changes 4-48
- data storage 4-78
 - for specific Column component 4-80
- data type
 - conversions A-2, A-4
- data type coercions A-3, A-4
 - caution for 4-33
 - from source to column 4-30
- data type mappings
 - overriding 4-33
- data types 2-6, A-1, A-3
 - caution for assigning 4-33
 - coercions 2-6
 - conversion A-1, A-3
- data validation
 - error handling 4-320
- DATA_CHANGE
 - AccessEvent class 4-11
 - StatusEvent class 4-271
- DATA_CHANGED
 - DataChangeEvent class 4-73
- DATA_FILE_LOAD_FAILED
 - DataSetException class 4-121
- DATA_HAS_DUPLICATES
 - DataStoreException class 2-37
- data-aware controls
 - associating with data sets 4-86
 - associating with pick lists 4-193
 - error handling 4-116
 - error-handling overrides 4-160
- database
 - ProcedureDataSet class 5-36
 - ProcedureResolver component 5-53
 - QueryDataSet class 5-61
 - QueryDescriptor class 5-75
 - QueryResolver component 5-83
 - SQLResolutionManager component 5-98
 - SQLResolver class 5-100
- Database component 5-13
- Database constructor
 - Database component 5-16
- databases 5-13
 - accessing SQL *See* SQL
 - connecting to *See* connections
- dataChange
 - DataSet class 4-113
 - DataSetView component 4-151
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-70
 - StorageDataSet component 4-301
 - TableDataSet component 4-314
- DataChangeAdapter class 4-72
 - methods 4-72
 - properties 4-72
- dataChanged
 - DataChangeListener interface 4-76
- DataChangeEvent class 4-72
- DataChangeEvent constructor
 - DataChangeEvent class 4-74
- DataChangeListener interface 4-76
 - adapter for 4-72
- DataExpress API 5-58
- DataExpress library overview 1-1
- dataFile
 - StorageDataSet component 4-278
- DataFile class 4-77
- DataFileFormat class 4-78
 - methods 4-79
 - properties 4-79
- dataLoaded
 - LoadListener interface 4-165
- DataModule interface 4-80
- DataRow class 4-80
 - properties 4-82
- DataRow constructor
 - DataRow class 4-81, 4-82

- dataSet
 - AggOperator class 4-21
 - Column component 4-38
 - ColumnVariant class 4-64
 - DataSetAware interface 4-114
 - ExceptionEvent class 4-161
 - ResolutionException class 5-88, 5-90
 - RowIterator class 4-254
- DataSet class 4-86
- dataset classes misc
 - overview in DataExpress library 4-9, 5-5
- dataset package
 - overview of classes 4-4
- DATASET_ALREADY_OPEN
 - DataStoreException class 2-38
- DATASET_CORRUPT
 - DataSetException class 4-121
- DATASET_EXISTS
 - DataStoreException class 2-38
- DATASET_HAS_NO_ROWS
 - DataSetException class 4-122
- DATASET_HAS_NO_TABLES
 - DataSetException class 4-122
- DATASET_NOT_OPEN
 - DataSetException class 4-122
- DATASET_OPEN
 - DataSetException class 4-122
- DataSetAware interface 4-114
- DataSetData class 4-114
 - properties 4-115
- DataSetException class 4-116
- DataSetException constructor
 - DataSetException class 4-133
- dataSetHasNoTable
 - DataSetException class 4-137
- dataSetNotOpen
 - DataSetException class 4-137
- datasets
 - overview in com.borland library 4-5
- DataSetView component 4-143
 - methods 4-146
- DataSetView constructor
 - DataSetView component 4-144
- dataStore
 - DataStoreConnection component 2-21
- DataStore component 2-3
 - data type coercions 2-6
 - specifications 2-7
- DataStore constructor
 - DataStore component 2-13
- datastore package
 - overview of classes 2-2
- datastore.jdbc package
 - overview of classes 3-2
- DATASTORE_ALREADY_OPEN
 - DataStoreException class 2-38
 - ResponseEvent class 4-243
- DATASTORE_CAN_REOPEN
 - ResponseEvent class 4-243
- DATASTORE_EXISTS
 - DataStoreException class 2-38
- DATASTORE_INVALID
 - DataStoreException class 2-38
- DATASTORE_NOT_FOUND
 - DataStoreException class 2-38
- DATASTORE_NOT_OPEN
 - DataStoreException class 2-38
- DATASTORE_OPEN
 - DataStoreException class 2-38
- DATASTORE_RECOVERING
 - ResponseEvent class 4-244
- DataStoreConnection component 2-19
- DataStoreConnection constructor
 - DataStoreConnection component 2-21
- DataStoreDriver component 3-2
 - methods 3-4
- DataStoreDriver constructor
 - DataStoreDriver component 3-4
- DataStoreException class 2-32
 - methods 2-43
 - properties 2-43
- DataStoreServer component 3-5
- DataStoreServer constructor
 - DataStoreServer component 3-6
- dataType
 - Column component 4-38
- DATE
 - Variant component 4-336
- date
 - Variant component 4-344
- date/time masks B-3
- DateFormatter component 6-23
- DateFormatter constructor
 - DateFormatter component 6-23
- DateType_S
 - Variant component 4-336
- debugging 5-15
- debugging JDBC connections 5-15
- DEFAULT
 - RowStatus interface 4-261
 - TriStateProperty interface 8-51
- default
 - Column component 4-39
- DEFAULT_CHECK_FREQUENCY
 - TxManager component 2-65
- DEFAULT_DRIVERS
 - Database component 5-16
- DEFAULT_HIDDEN
 - RowStatus interface 4-261

- DEFAULT_LOG_BLOCK_SIZE
 - TxManager component 2-65
- DEFAULT_LOG_SIZE
 - TxManager component 2-65
- DEFAULT_OPEN_LOGS
 - TxManager component 2-65
- DEFAULT_PORT
 - DataStoreServer component 3-6
- defaultEventIndex
 - BasicBeanInfo class 8-8, 8-12
- defaultPropertyIndex
 - BasicBeanInfo class 8-8, 8-12
- DefaultResolver interface 5-28
- defaults
 - display masks 4-39
- defaultValue
 - Column component 4-39
- defaultValues
 - DataSet class 4-88
- delete
 - AggOperator class 4-22
 - CountAggOperator class 4-67
 - CustomAggOperator class 4-69
 - ItemEditMask interface 6-33
 - SumAggOperator class 4-305
- DELETE_DUPLICATES
 - DataSetException class 4-122
- DELETE_FAILED
 - ResolutionException class 5-89
- DELETE_NOT_ALLOWED
 - ValidationException class 4-328
- deleteAllRows
 - DataSet class 4-100
- DELETED
 - RowStatus interface 4-262
- deleted
 - EditListener interface 4-156
- DELETED_STATE
 - DataStore component 2-9
- DELETED_STREAM
 - DataStore component 2-9
- deletedRow
 - ResolverListener interface 4-236
- deletedRowCount
 - StorageDataSet component 4-279
- deleteDuplicates
 - DataSetException class 4-137
 - StorageDataSet component 4-293
- deleteError
 - EditListener interface 4-157
 - ResolverListener interface 4-236
- deleteProcedure
 - ProcedureResolver component 5-53
- deleteRow
 - DataSet class 4-100
- ProcedureResolver component 5-55
- QueryResolver component 5-84
- RowIterator class 4-259
- SQLResolver class 5-101
- deleteStream
 - DataStoreConnection component 2-27
- deleting
 - EditListener interface 4-157
- deletingRow
 - ResolverListener interface 4-237
- delimiter
 - TextDataFile component 4-316
- descending
 - SortDescriptor class 4-267
 - StreamProperties class 2-49
- descriptor classes
 - AggDescriptor 4-17
 - ConnectionDescriptor 5-5
 - MasterLinkDescriptor 4-168
 - PickListDescriptor 4-193
 - SortDescriptor 4-263
- descriptors for datasets
 - overview in DataExpress library 4-6, 5-4
- designing applications 5-14
- destinationColumns
 - PickListDescriptor class 4-196
- DETAIL
 - Locate interface 4-167
- detail queries 4-170
- detail tables 4-168
 - getting values from 4-168
 - specifying master for 4-168
- detailDataSetWithFetchAsNeeded
 - DataSet class 4-88
- detailLinkColumns
 - MasterLinkDescriptor class 4-174
- details
 - DataSet class 4-88
- developer support 1-3
- Diagnostic class 8-14
 - properties 8-15
- diagnostic classes
 - overview 8-2
- DIR_ACCESS
 - DataStore component 2-9
- DIR_BLOB_LENGTH
 - DataStore component 2-9
- DIR_DEL_TIME
 - DataStore component 2-10
- DIR_ID
 - DataStore component 2-10
- DIR_LENGTH
 - DataStore component 2-10
- DIR_MOD_TIME
 - DataStore component 2-10

- DIR_PROPERTIES
 - DataStore component 2-10
- DIR_STATE
 - DataStore component 2-10
- DIR_STORE_NAME
 - DataStore component 2-10
- DIR_TYPE
 - DataStore component 2-11
- DISCONNECT
 - ServerStatus interface 3-12
- dispatch
 - AccessEvent class 4-16
 - DataChangeEvent class 4-75
 - DispatchableEvent class 8-24
 - EventMulticaster class 8-29
 - ExceptionEvent class 4-161
 - LoadEvent class 4-164
 - NavigationEvent class 4-182
 - ResponseEvent class 4-248
 - ServerStatusEvent class 3-13
 - StatusEvent class 4-274
- DispatchableEvent class 8-22
 - variables 8-22
- DispatchableEvent constructor
 - DispatchableEvent class 8-22
- display masks B-1
 - setting as default 4-39
 - storing format specifications 4-39
- displayErrors
 - DataSet class 4-88
- displaying data 4-250
- displayMask
 - Column component 4-39
- displayString
 - ItemEditMaskState component 6-36
- displayValue
 - Variant component 4-344
- dittoRow
 - DataSet class 4-100, 4-101
- documentation
 - printing conventions 1-3
- documentation conventions 1-3
- doTransactions
 - SQLResolutionManager component 5-98
- DOUBLE
 - Variant component 4-336
- double
 - Variant component 4-345
- DoubleFormatter class 6-25
- DoubleFormatter constructor
 - DoubleFormatter class 6-25
- DoubleType_S
 - Variant component 4-336
- driver
 - ConnectionDescriptor class 5-7

- DRIVER_NOT_LOADED_AT_RUNTIME
 - DataSetException class 4-122
- DRIVER_NOT_LOADED_IN_DESIGN
 - DataSetException class 4-122
- driverNotLoadedAtRuntime
 - DataSetException class 4-137
- driverNotLoadedInDesign
 - DataSetException class 4-137
- DROP_LOG
 - ResponseEvent class 4-244
- dropAllIndexes
 - StorageDataSet component 4-293
- dropColumn
 - AccessEvent class 4-15
 - StorageDataSet component 4-293
- dropIndex
 - DataSet class 4-101
 - StorageDataSet component 4-293
- dropIndexStream
 - DataStoreConnection component 2-27
- dumpLocks
 - DataStoreConnection component 2-27
- dumpTxLog
 - DataStoreConnection component 2-27
- DUPLICATE_COLUMN_NAME
 - DataSetException class 4-123
- DUPLICATE_KEY
 - DataStoreException class 2-39
 - ValidationException class 4-328
- DUPLICATE_PRIMARY
 - DataSetException class 4-123
- duplicateKey
 - ValidationException class 4-333
- duplicates
 - StorageDataSet component 4-279
- dx.sql.dataset package
 - overview of classes 5-3
- dx.text package
 - overview of classes 6-2
- dxDispatch
 - DxDispatch interface 4-152
- DxDispatch interface 4-151

E

- edit
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-70
 - StorageDataSet component 4-301
 - TableDataSet component 4-314
- edit masks 4-41, B-1
- edit notifications 4-152, 4-153
- EDIT_CANCELED
 - StatusEvent class 4-271
- EDIT_STARTED
 - StatusEvent class 4-271

- editable
 - Column component 4-40
 - DataSet class 4-88
- EditAdapter class 4-152
 - methods 4-152
 - properties 4-152
- editError
 - EditListener interface 4-157
- editing 4-170, B-1
 - ColumnPaintListener interface 4-61
 - DataSet class 4-88
 - RowIterator class 4-254
- editing data
 - adapter for JDBC objects 4-152
- editing properties 4-114
- editingNewRow
 - DataSet class 4-89
 - RowIterator class 4-254
- EditListener interface 4-153
 - adapter for 4-152
- editMask
 - Column component 4-41
- editMasker
 - Column component 4-41
- editRow
 - DataSet class 4-101
- empty
 - DataSet class 4-89
 - FastStringBuffer component 8-40
 - StorageDataSet component 4-294
- EMPTY_COLUMN_NAMES
 - DataSetException class 4-123
- emptyAllRows
 - DataSet class 4-101
- emptyRow
 - DataSet class 4-102
- enableChecking
 - Diagnostic class 8-17
- enabled
 - TxManager component 2-68
- enableDataSetEvents
 - DataSet class 4-102
- enableDelete
 - DataSet class 4-89
- enableInsert
 - DataSet class 4-89
- enableOutput
 - Diagnostic class 8-17
- enableUpdate
 - DataSet class 4-89
- ENCODED
 - DataFileFormat class 4-79
- EncodedInputStream class 7-7
 - properties 7-7
- EncodedInputStream constructor
 - EncodedInputStream class 7-7
- EncodedOutputStream class 7-9
 - properties 7-10
- EncodedOutputStream constructor
 - EncodedOutputStream class 7-9
- encoding
 - TextDataFile component 4-316
- endLoading
 - StorageDataSet component 4-294
- endResolution
 - ProviderHelp class 4-202
- enforceIntegrity
 - PickListDescriptor class 4-196
- ENUMERATION
 - BasicBeanInfo class 8-8
- equals
 - ReadRow class 4-208
 - SortDescriptor class 4-269
 - Variant component 4-349
- equalsInstance
 - Variant component 4-349
- error
 - ServerStatusEvent class 3-12
- error constants 5-85
- errorCode
 - DataSetException class 4-123, 4-134
- errorColumn
 - ValidationException class 4-330
- errorDataSets
 - ResolutionException class 5-91
- errorOffset
 - InvalidFormatException class 6-31
- ErrorResponse component 8-24
- ErrorResponse constructor
 - ErrorResponse component 8-25
- errors
 - See also* exceptions
 - column and row validation 4-320
 - handling common data set 4-116
 - resolving data changes
- event classes for datasets
 - overview in DataExpress library 4-6, 5-3
- event-dispatching classes
 - overview 8-2
- eventListenerMethods
 - BasicBeanInfo class 8-9
- EventMulticaster class 8-27
- events
 - for dataset component writers 4-7
- eventSetDescriptors
 - BasicBeanInfo class 8-9, 8-12
- ex
 - ResolveError class 5-95

EXCEPTION

- StatusEvent class 4-271
- StreamVerifier class 2-51
- exception
 - ExceptionChain component 8-31
 - ExceptionEvent class 4-161
 - ExceptionListener interface 4-162
 - ResponseEvent class 4-247
 - StatusEvent class 4-273
- EXCEPTION_CHAIN
 - DataSetException class 4-123
- exceptionChain
 - ChainedException interface 8-14
 - DataSetException class 4-123, 4-134
 - DispatchableEvent class 8-23
- ExceptionChain component 8-30
- ExceptionChain constructor
 - ExceptionChain component 8-31
- exceptionDispatch
 - EventMulticaster class 8-29
 - ExceptionDispatch interface 8-33
- ExceptionDispatch interface 8-32
- ExceptionEvent class 4-160
 - variables 4-160
- ExceptionEvent constructor
 - ExceptionEvent class 4-160
- ExceptionHandler interface 8-33
- ExceptionListener interface 4-162
- exception-related classes
 - overview 8-2
 - overview in DataExpress library 6-4
- exceptions
 - data validation 4-320
 - data-aware overrides 4-160
 - dataset packages 4-116
 - listener for 4-162
 - registering listeners 4-162
- exceptions for datasets
 - overview in com.borland library 4-8, 5-4
- executeOnOpen
 - QueryDescriptor class 5-75
- executeQuery
 - ProcedureDataSet class 5-43
 - QueryDataSet class 5-68
- executeStatement
 - Database component 5-24, 5-72
 - QueryProvider component 5-80
- exit
 - Diagnostic class 8-17
- expandDelimiters
 - FastStringBuffer component 8-40
- export masks B-1
- exportDisplayMask
 - Column component 4-42

- exportFormatter
 - Column component 4-43
- exporting data 4-77
 - to text files 4-315
- extractDataSet
 - DataSetData class 4-115
- extractDataSetChanges
 - DataSetData class 4-116

F

- fail
 - Diagnostic class 8-17
- failIfOpen
 - ProviderHelp class 4-202
- FALSE
 - TriStateProperty interface 8-52
- falseString
 - BooleanFormat component 6-16
- FAST
 - Locate interface 4-167
- FastBufferedInputStream class 7-11
 - properties 7-12
 - variables 7-11
- FastBufferedInputStream constructor
 - FastBufferedInputStream class 7-11, 7-12
- FastBufferedOutputStream class 7-15
 - properties 7-16
- FastBufferedOutputStream constructor
 - FastBufferedOutputStream class 7-15, 7-16
- FastStringBuffer component 8-33
- FastStringBuffer constructor
 - FastStringBuffer component 8-34, 8-35
- FETCH_STREAM
 - DataStore component 2-11
- fetchAsNeeded
 - editing operations 4-170
 - MasterLinkDescriptor class 4-174
- fetching detail records 4-168
- fetchResolverListener
 - SQLResolver class 5-101
- FIELD_POST_ERROR
 - DataSetException class 4-123
- FILE_EXISTS
 - DataStoreException class 2-39
 - ResponseEvent class 4-244
- FILE_STREAM
 - DataStore component 2-11
- fileExists
 - DataStoreConnection component 2-27
- fileFormat
 - TextDataFile component 4-317
- fileLength
 - DataStore component 2-18
- fileName
 - DataStoreConnection component 2-21

- TextDataFile component 4-317
- FileOutputStream class 2-45
 - properties 2-45
- files 4-315
 - importing from data-based 4-305
- FileStream class 2-46
 - properties 2-47
- fill
 - FastBufferedInputStream class 7-13
- fillCharacter
 - TextFormat component 6-66
- filterClassName
 - StreamProperties class 2-49
- filtering data 4-144, 4-250
- filterRow
 - RowFilterListener interface 4-250
- finalize
 - Database component 5-24
 - DataStore component 2-18
 - DataStoreConnection component 2-28
- find
 - EventMulticaster class 8-29
- findDifference
 - ReadRow class 4-208
- finding data *See* searching data
- findModified
 - ReadRow class 4-208
- findOrdinal
 - ColumnList component 4-59
 - ReadRow class 4-208
- fire
 - InvalidFormatException class 6-32
 - VariantException class 4-353
- FIRST
 - Locate interface 4-167
- first
 - DataSet class 4-102
 - RowIterator class 4-259
- firstChar
 - FastStringBuffer component 8-40
- fixedPrecision
 - Column component 4-43
- FLOAT
 - Variant component 4-337
- float
 - Variant component 4-345
- FloatType_S
 - Variant component 4-337
- flush
 - AsciiOutputStream class 7-6
 - Diagnostic class 8-18
 - EncodedOutputStream class 7-10
 - FastBufferedOutputStream class 7-16
- font
 - Column component 4-43

- CustomPaintSite interface 4-70
- foreground
 - Column component 4-43
 - CustomPaintSite interface 4-71
- format
 - BigDecimalFormatter class 6-9
 - BinaryFormatter component 6-12
 - BooleanFormat component 6-17, 6-18
 - BooleanFormatter component 6-20
 - Column component 4-51
 - DateFormatter component 6-25
 - DoubleFormatter class 6-27
 - IntegerFormatter class 6-29
 - ItemFormatStr class 6-46
 - ItemFormatter class 6-49
 - LongFormatter component 6-52
 - ObjectFormatter component 6-54
 - ReadRow class 4-209
 - SimpleFormatter component 6-60
 - StringFormatter component 6-63
 - TextFormat component 6-68
 - TimeFormatter component 6-71
 - TimestampFormatter component 6-73
 - VariantFormatStr class 6-78
 - VariantFormatter class 6-82, 6-83
- format specifiers
 - storing 4-39
- formatObj
 - BinaryFormatter component 6-10
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-48
 - SimpleFormatter component 6-59
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- formatter
 - Column component 4-43
 - displayMask as 4-39
- formatting B-1

G

- GENERIC_ERROR
 - DataSetException class 4-123
- getAdditionalBeanInfo
 - BasicBeanInfo class 8-11
- getAgg
 - Column component 4-36
- getAggColumnName
 - AggDescriptor class 4-19
- getAggOperator
 - AggDescriptor class 4-19
- getAlignment
 - Column component 4-36
 - CustomPaintSite interface 4-70
- getALogDir
 - TxManager component 2-67

- getArrayLength
 - ReadRow class 4-209
 - Variant component 4-341
- getAsBigDecimal
 - Variant component 4-341
- getAsBoolean
 - Variant component 4-341
- getAsDouble
 - Variant component 4-342
- getAsFloat
 - Variant component 4-342
- getAsInt
 - Variant component 4-342
- getAsLong
 - Variant component 4-342
- getAsObject
 - Variant component 4-342
- getAsShort
 - Variant component 4-343
- getAutoCommit
 - Database component 5-17
- getBackground
 - Column component 4-36
 - CustomPaintSite interface 4-70
- getBeanDescriptor
 - BasicBeanInfo class 8-12
- getBigDecimal
 - ReadRow class 4-209
 - Variant component 4-344
- getBinaryStream
 - ReadRow class 4-209
 - Variant component 4-344
- getBlockSize
 - DataStore component 2-14
- getBLogDir
 - TxManager component 2-67
- getBoolean
 - ReadRow class 4-210
 - Variant component 4-344
- getByte
 - ReadRow class 4-210
 - Variant component 4-344
- getByteArray
 - ReadRow class 4-210, 4-211
 - Variant component 4-344
- getBytes
 - InputStreamToByteArray class 7-18, 7-19
- getCalcAggFieldsListener
 - StorageDataSet component 4-278
- getCalcFieldsListener
 - StorageDataSet component 4-278
- getCalcType
 - Column component 4-36
- getCaption
 - Column component 4-37
- getChars
 - FastStringBuffer component 8-40
- getCheckFrequency
 - TxManager component 2-68
- getCode
 - ResponseEvent class 4-247
 - StatusEvent class 4-273
- getColumn
 - ColumnList component 4-59
 - ColumnVariant class 4-64
 - ReadRow class 4-211
- getColumnChangeListener
 - Column component 4-37
- getColumnCount
 - ReadRow class 4-205
- columnName
 - Column component 4-37
 - ColumnAware interface 4-54
- getColumnNames
 - ReadRow class 4-211
- getColumnPaintListener
 - Column component 4-37
- getColumnns
 - ParameterRow component 4-186
 - ReadRow class 4-205
 - StreamProperties class 2-48
- getColumnnsArray
 - ColumnList component 4-58
- GetComponent
 - ExceptionEvent class 4-161
- getConnection
 - Database component 5-17
- getConnectionURL
 - ConnectionDescriptor class 5-7
- getContents
 - ArrayResourceBundle class 8-5
- getDatabase
 - ProcedureDataSet class 5-36
 - ProcedureResolver component 5-53
 - QueryDataSet class 5-61
 - QueryDescriptor class 5-75
 - QueryResolver component 5-83
 - SQLResolutionManager component 5-98
 - SQLResolver class 5-100
- getDataFile
 - StorageDataSet component 4-278
- getDataRow
 - DataSet class 4-102
- getDataSet
 - Column component 4-38
 - ColumnVariant class 4-64
 - DataSetAware interface 4-114
 - ExceptionEvent class 4-161
 - ResolutionException class 5-90
 - RowIterator class 4-254

- getDataSource
 - DataSourceConnection component 2-21
- getDataType
 - Column component 4-38
- getDate
 - ReadRow class 4-211
 - Variant component 4-344
- getDefault
 - Column component 4-39, 4-52
- getDefaultEventIndex
 - BasicBeanInfo class 8-12
- getDefaultIconResource
 - BasicBeanInfo class 8-13
- getDefaultPattern
 - VariantFormatStr class 6-78
- getDefaultPropertyIndex
 - BasicBeanInfo class 8-12
- getDefaultValue
 - Column component 4-39
- getDeletedRowCount
 - StorageDataSet component 4-279
- getDeletedRows
 - StorageDataSet component 4-294
- getDeleteProcedure
 - ProcedureResolver component 5-53
- getDelimiter
 - TextDataFile component 4-316
- getDescending
 - SortDescriptor class 4-267
 - StreamProperties class 2-49
- getDestinationColumns
 - PickListDescriptor class 4-196
- getDetail
 - DataSet class 4-103
- getDetailLinkColumns
 - MasterLinkDescriptor class 4-174
- getDetails
 - DataSet class 4-88
- getDisplayMask
 - Column component 4-39
- getDisplayValue
 - Variant component 4-344
- getDisplayVariant
 - DataSet class 4-103
- getDouble
 - ReadRow class 4-212
 - Variant component 4-345
- getDriver
 - ConnectionDescriptor class 5-7
- getDropColumn
 - AccessEvent class 4-15
- getDuplicates
 - StorageDataSet component 4-279
- getEditMask
 - Column component 4-41
- getEditMasker
 - Column component 4-41
- getEncoding
 - TextDataFile component 4-316
- getError
 - ServerStatusEvent class 3-12
- getErrorCode
 - DataSetException class 4-134
- getErrorColumn
 - ValidationException class 4-330
- getErrorDataSets
 - ResolutionException class 5-91
- getErrorOffset
 - InvalidFormatException class 6-31
- getEventSetDescriptors
 - BasicBeanInfo class 8-12
- getException
 - ExceptionChain component 8-31
 - ExceptionEvent class 4-161
 - ResponseEvent class 4-247
 - StatusEvent class 4-273
- getExceptionChain
 - ChainedException interface 8-14
 - DataSetException class 4-134
 - DispatchableEvent class 8-23
- getExceptionListeners
 - DataSetException class 4-137
- getExportDisplayMask
 - Column component 4-42
- getExportFormatter
 - Column component 4-43
- getFalseString
 - BooleanFormat component 6-16
- getFile
 - SearchPath class 8-51
- getFileFormat
 - TextDataFile component 4-317
- getFileName
 - DataSourceConnection component 2-21
 - TextDataFile component 4-317
- getFillCharacter
 - TextFormat component 6-66
- getFilterClassName
 - StreamProperties class 2-49
- getFinalValue
 - ItemEditMask interface 6-33, 6-34
- getFloat
 - ReadRow class 4-212
 - Variant component 4-345
- getFont
 - Column component 4-43
 - CustomPaintSite interface 4-70
- getForeground
 - Column component 4-43
 - CustomPaintSite interface 4-71

- getFormatObj
 - BinaryFormatter component 6-10
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-48
 - SimpleFormatter component 6-59
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- getFormatter
 - Column component 4-43
- getGroupColumnNames
 - AggDescriptor class 4-19
- getID
 - AccessEvent class 4-15
 - DataChangeEvent class 4-74
- getIdentifierQuoteChar
 - Database component 5-17
- getImage
 - BasicBeanInfo class 8-13
- getIndexName
 - SortDescriptor class 4-267
 - StreamProperties class 2-49
- getInputStream
 - ReadRow class 4-212, 4-213
 - Variant component 4-345
- getInsertedRowCount
 - StorageDataSet component 4-279
- getInsertedRows
 - StorageDataSet component 4-294
- getInsertProcedure
 - ProcedureResolver component 5-53
- getInt
 - ReadRow class 4-213
 - Variant component 4-345
- getInternalRow
 - DataSet class 4-90
 - RowIterator class 4-254
- getItemEditor
 - Column component 4-44
- getItemMargins
 - CustomPaintSite interface 4-71
- getItemPainter
 - Column component 4-44
- getJavaClass
 - Column component 4-44
- getJdbcConnection
 - Database component 5-17
- getKeepLiterals
 - TextFormat component 6-67
- getKeys
 - ArrayResourceBundle class 8-5
 - SortDescriptor class 4-267
- getLastColumnVisited
 - DataSet class 4-90
- getLength
 - FastStringBuffer component 8-36
- getListenerCount
 - EventMulticaster class 8-28
- getLoadOption
 - QueryDescriptor class 5-76
- getLocale
 - BinaryFormatter component 6-10
 - Column component 4-44
 - DataStore component 2-14
 - DataStoreConnection component 2-22
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-48
 - LocaleUtil class 8-49
 - SimpleFormatter component 6-59
 - SortDescriptor class 4-267
 - StorageDataSet component 4-279
 - TextDataFile component 4-317
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- getLocaleName
 - SortDescriptor class 4-267
- getLockWaitTime
 - DataStoreConnection component 2-22
- getLogBlockSize
 - TxManager component 2-68
- getLong
 - ReadRow class 4-213, 4-214
 - Variant component 4-345
- getLookupDisplayColumn
 - PickListDescriptor class 4-197
- getMajorVersion
 - DataStoreDriver component 3-4
- getMasterDataSet
 - MasterLinkDescriptor class 4-175
- getMasterLink
 - DataSet class 4-90
 - DataSetView component 4-145
- getMasterLinkColumns
 - MasterLinkDescriptor class 4-175
- getMax
 - Column component 4-45
- getMaxDesignRows
 - StorageDataSet component 4-279
- getMaxInline
 - Column component 4-45
- getMaxLogSize
 - TxManager component 2-68
- getMaxOpenLogs
 - TxManager component 2-68
- getMaxResolveErrors
 - StorageDataSet component 4-280
- getMaxRowLength
 - DataStore component 2-14
- getMaxRowLocks
 - DataStore component 2-14

- getMaxRows
 - StorageDataSet component 4-280
- getMaxSortBuffer
 - DataStore component 2-15
- getMaxStatements
 - Database component 5-18
- getMaxValue
 - Column component 4-45
- getMessage
 - ResponseEvent class 4-247
 - StatusEvent class 4-274
- getMetaData
 - Database component 5-18
- getMetaDataUpdate
 - StorageDataSet component 4-280
- getMethodDescriptors
 - BasicBeanInfo class 8-12
- getMin
 - Column component 4-45
- getMinCacheSize
 - DataStore component 2-15
- getMinorVersion
 - DataStoreDriver component 3-4
- getMinValue
 - Column component 4-46
- getNeedsRestructure
 - StorageDataSet component 4-281
- getNewColumn
 - AccessEvent class 4-15
- getNewOrdinal
 - AccessEvent class 4-15
- getNext
 - ExceptionChain component 8-31
- getNullString
 - BooleanFormat component 6-16
- getObject
 - ArrayResourceBundle class 8-6
 - ReadRow class 4-214
 - Variant component 4-346
- getOffset
 - FastStringBuffer component 8-36
- getOldColumn
 - AccessEvent class 4-16
- getOldOrdinal
 - AccessEvent class 4-16
- getOpenMode
 - DataStore component 2-15
- getOptions
 - SortDescriptor class 4-268
- getOrdinal
 - Column component 4-46
- getOriginalMessage
 - ExceptionChain component 8-32
- getOriginalQueryString
 - QueryDataSet class 5-61
- getOriginalRow
 - StorageDataSet component 4-294
- getParameterRow
 - ProcedureDataSet class 5-36
 - ProcedureProvider class 5-49
 - Provider class 4-198
 - QueryDataSet class 5-61
 - QueryDescriptor class 5-76
 - QueryProvider component 5-78
- getParameterType
 - Column component 4-46
- getPassword
 - ConnectionDescriptor class 5-8
- getPath
 - SearchPath class 8-51
- getPattern
 - BinaryFormatter component 6-11
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-49
 - SimpleFormatter component 6-59
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- getPickList
 - Column component 4-47
- getPickListColumns
 - PickListDescriptor class 4-197
- getPickListDataSet
 - PickListDescriptor class 4-197
- getPickListDisplayColumns
 - PickListDescriptor class 4-197
- getPort
 - DataStoreServer component 3-6
- getPrecision
 - Column component 4-47
- getPreferredOrdinal
 - Column component 4-47
- getProcedure
 - ProcedureDataSet class 5-36
 - ProcedureProvider class 5-49
- getProductVersion
 - DataStoreConnection component 2-28
- getProperties
 - ConnectionDescriptor class 5-8
- getPropertyDescriptors
 - BasicBeanInfo class 8-12
- getProvider
 - StorageDataSet component 4-281
- getQuery
 - QueryDataSet class 5-62
 - QueryProvider component 5-78
- getQueryString
 - ProcedureDataSet class 5-36
 - QueryDataSet class 5-69
 - QueryDescriptor class 5-76
 - QueryProvider component 5-81

- getReadOnlyTxDelay
 - DataStoreConnection component 2-23
- getReadRow
 - RowIterator class 4-254
- getReadWriteRow
 - RowIterator class 4-254
- getReason
 - AccessEvent class 4-16
- getReplaceCharacter
 - TextFormat component 6-67
- getResolveOrder
 - StorageDataSet component 4-282
- getResolver
 - DefaultResolver interface 5-28
 - StorageDataSet component 4-282
- getResolverDataSet
 - ProviderHelp class 4-202
- getResolverQueryTimeout
 - QueryResolver component 5-83
- getResponse
 - ErrorResponse component 8-26
 - ResponseEvent class 4-247
- getRow
 - DataSet class 4-90
 - RowIterator class 4-254
- getRowAffected
 - DataChangeEvent class 4-74
- getRowCount
 - DataSet class 4-91
- getRowFilterListener
 - DataSet class 4-91
- getRuntimeMetaData
 - Database component 5-18
- getSaveInterval
 - DataStore component 2-15
- getSaveMode
 - DataStore component 2-16
- getScale
 - Column component 4-48
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- getSchemaName
 - Column component 4-49
 - DataSet class 4-91
 - StorageDataSet component 4-283
- getSeparator
 - TextDataFile component 4-317
- getServerColumnName
 - Column component 4-49
- getServerConnection
 - ServerStatusEvent class 3-13
- getServerConnections
 - DataStoreServer component 3-7
- getSetType
 - Variant component 4-346
- getShort
 - ReadRow class 4-214
 - Variant component 4-346
- getSiteComponent
 - CustomPaintSite interface 4-71
- getSort
 - DataSet class 4-91
- getSortPrecision
 - Column component 4-49
- getSpecialObject
 - BinaryFormatter component 6-12
 - ItemFormatStr class 6-46
 - ItemFormatter class 6-49
 - SimpleFormatter component 6-60
 - VariantFormatStr class 6-78
 - VariantFormatter class 6-83
- getSQLDialect
 - Database component 5-18
- getSqlType
 - Column component 4-50
- getStatus
 - DataSet class 4-91
- getStatusCode
 - ServerStatusEvent class 3-13
- getStorageDataSet
 - DataSet class 4-92
 - StoreClassFactory interface 4-303
- getStore
 - StorageDataSet component 4-283
- getStoreClassFactory
 - StorageDataSet component 4-283
- getStoreInternals
 - DataStoreConnection component 2-23
- getStoreName
 - StorageDataSet component 4-283
- getStreamProperties
 - DataStoreConnection component 2-28
- getString
 - ArrayResourceBundle class 8-6
 - ReadRow class 4-215
 - Variant component 4-346
- getStringArray
 - ArrayResourceBundle class 8-6
- getStructureAge
 - ProviderHelp class 4-202
- getTableName
 - Column component 4-50
 - DataSet class 4-92
 - StorageDataSet component 4-284
- getTempDir
 - DataStoreServer component 3-7
- getTempDirName
 - DataStore component 2-16
- getTime
 - ReadRow class 4-215

- Variant component 4-347
- getTimestamp
 - ReadRow class 4-216
 - Variant component 4-347
- getTimeZoneOffset
 - Variant component 4-350
- getTraceLevel
 - Diagnostic class 8-18
- getTransactionIsolation
 - Database component 5-18
- getTrueString
 - BooleanFormat component 6-16
- getTxIsolation
 - DataStoreConnection component 2-23
- getTxManager
 - DataStore component 2-16
- getType
 - Variant component 4-347
- getUpdatedRowCount
 - StorageDataSet component 4-284
- getUpdatedRows
 - StorageDataSet component 4-294
- getUpdateMode
 - QueryResolver component 5-83
- getUpdateProcedure
 - ProcedureResolver component 5-53
- getUserName
 - ConnectionDescriptor class 5-8
 - DataStoreConnection component 2-23
 - ServerConnection class 3-10
- getValue
 - FastStringBuffer component 8-36
- getVariant
 - DataSet class 4-103
 - ReadRow class 4-216
- getVariantType
 - BigDecimalFormatter class 6-8
 - BinaryFormatter component 6-11
 - BooleanFormatter component 6-19
 - DateFormatter component 6-24
 - DoubleFormatter class 6-26
 - IntegerFormatter class 6-28
 - LongFormatter component 6-51
 - ObjectFormatter component 6-53
 - SimpleFormatter component 6-59
 - StringFormatter component 6-62
 - TimeFormatter component 6-70
 - TimestampFormatter component 6-72
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- getVersion
 - DataStoreConnection component 2-23
- getVetoMessage
 - VetoException component 8-53

- getVisible
 - Column component 4-50
- getWidth
 - Column component 4-50
- goToClosestRow
 - DataSet class 4-104
- goToInternalRow
 - DataSet class 4-104
- goToRow
 - DataSet class 4-104
- GREATER_THAN_MAX
 - ValidationException class 4-328
- groupColumnNames
 - AggDescriptor class 4-19

H

- handleException
 - ExceptionHandler interface 8-33
- handleGetObject
 - ArrayResourceBundle class 8-6
- hasColumn
 - ColumnList component 4-59
 - ReadRow class 4-216
- hasDetail
 - DataSet class 4-104
- hasExceptions
 - ExceptionChain component 8-32
- hasListeners
 - EventMulticaster class 8-30
- hasMoreData
 - Provider class 4-200
- hasRowIds
 - StorageDataSet component 4-295
- hasValidations
 - Column component 4-52
 - DataSet class 4-105
- Hex class 8-47
 - methods 8-48
 - properties 8-48
- hidden
 - Column component 4-44
- HIDDEN_STREAM
 - DataStore component 2-11
- HORIZONTAL
 - Alignment class 6-5
- HSTRETCH
 - Alignment class 6-6

I

- I/O classes
 - overview 8-2
- iconColor16x16
 - BasicBeanInfo class 8-9

- iconColor32x32
 - BasicBeanInfo class 8-9
- iconMono16x16
 - BasicBeanInfo class 8-9
- iconMono32x32
 - BasicBeanInfo class 8-9
- ID
 - AccessEvent class 4-15
 - DataChangeEvent class 4-74
- identifierQuoteChar
 - Database component 5-17
- ifBusy
 - ProcedureProvider class 5-51
- IGNORE
 - ErrorResponse component 8-25
- ignore
 - ErrorResponse component 8-26, 8-27
 - RowFilterResponse class 4-252
- IGNORE_ALL
 - ResponseEvent class 4-244
- ignoreAll
 - ResponseEvent class 4-247, 4-248
- import masks B-1
- Import/export
 - overview in DataExpress library 4-8
- importing data 4-77, 4-305
 - from text files 4-305, 4-315
- IN
 - ParameterType class 4-192
- IN_OUT
 - ParameterType class 4-192
- inBounds
 - DataSet class 4-105
 - RowIterator class 4-259
- INCOMPATIBLE_BLOCK_SIZE
 - DataStoreException class 2-39
- INCOMPATIBLE_DATA_ROW
 - DataSetException class 4-124
- indexExists
 - StorageDataSet component 4-295
- indexName
 - SortDescriptor class 4-267
 - StreamProperties class 2-49
- indexOf
 - FastStringBuffer component 8-41
- IndexOfSubString
 - FastStringBuffer component 8-41
- init
 - AggOperator class 4-22
 - CountAggOperator class 4-67
- initData
 - ProviderHelp class 4-202, 4-203
- initError
 - SQLResolutionManager component 5-99

- INLINE_TOO_SMALL
 - DataStoreException class 2-39
- input
 - parsing user 4-39
 - restricting user 4-41
- INPUTSTREAM
 - Variant component 4-337
- inputStream
 - Variant component 4-345
- InputStreamToByteArray class 7-17
 - variables 7-17
- InputStreamToByteArray constructor
 - InputStreamToByteArray class 7-18
- InputStreamType_S
 - Variant component 4-337
- insert
 - FastStringBuffer component 8-41, 8-42
 - ItemEditMask interface 6-34
- INSERT_FAILED
 - ResolutionException class 5-89
- INSERT_NOT_ALLOWED
 - ValidationException class 4-328
- INSERTED
 - RowStatus interface 4-262
- inserted
 - EditListener interface 4-158
- INSERTED_STREAM
 - DataStore component 2-11
- insertedRow
 - ResolverListener interface 4-237
- insertedRowCount
 - StorageDataSet component 4-279
- insertError
 - ResolverListener interface 4-237
- inserting
 - EditListener interface 4-158
- insertingRow
 - ResolverListener interface 4-238
- insertProcedure
 - ProcedureResolver component 5-53
- insertRow
 - DataSet class 4-105
 - ProcedureResolver component 5-55
 - QueryResolver component 5-85
 - RowIterator class 4-259
 - SQLResolver class 5-102
- installation support 1-3
- INSUFFICIENT_ROWID
 - DataSetException class 4-124
- insufficientRowId
 - DataSetException class 4-138
- INT
 - Variant component 4-337
- int
 - Variant component 4-345

- IntegerFormatter class 6-27
- IntegerFormatter constructor
 - IntegerFormatter class 6-28
- interactiveLocate
 - DataSet class 4-106
- INTERBASE
 - SQLDialect interface 5-96
- interfaces (com.borland)
 - ServerStatus 3-11
 - ServerStatusListener 3-13
- internalRow
 - DataSet class 4-90
 - ResolveError class 5-95
 - RowIterator class 4-254
- internationalization classes
 - overview 8-3
- IntType_S
 - Variant component 4-337
- INVALID_AGG_DESCRIPTOR
 - DataSetException class 4-124
- INVALID_ANCHOR_FILE
 - TxException class 2-58
- INVALID_ANCHOR_LENGTH
 - TxException class 2-58
- INVALID_CLASS
 - DataSetException class 4-124
- INVALID_COLUMN_POSITION
 - DataSetException class 4-124
- INVALID_COLUMN_TYPE
 - DataSetException class 4-124
- INVALID_COLUMN_VALUE
 - ValidationException class 4-328
- INVALID_DATA_FILE_FORMAT
 - DataSetException class 4-124
- INVALID_DIRECTORY_ATTRIBUTES
 - DataStoreException class 2-39
- INVALID_FORMAT
 - DataSetException class 4-124
 - ValidationException class 4-328
- INVALID_ITERATOR_USE
 - DataSetException class 4-125
- INVALID_LSN
 - TxException class 2-58
- INVALID_PATTERN
 - DataStoreException class 2-39
- INVALID_PRECISION
 - ValidationException class 4-329
- INVALID_ROW_VALUES
 - ValidationException class 4-329
- INVALID_SCHEMA_FILE
 - DataSetException class 4-125
- INVALID_SORT_COLUMN
 - DataSetException class 4-125
- INVALID_STORE_CLASS
 - DataSetException class 4-125
- INVALID_STORE_NAME
 - DataSetException class 4-125
- INVALID_USER_NAME
 - TxException class 2-58
- invalidClass
 - DataSetException class 4-138
- invalidColumnType
 - DataSetException class 4-138
- invalidFormat
 - ValidationException class 4-333
- InvalidFormatException class 6-30
- InvalidFormatException constructor
 - InvalidFormatException class 6-31
- invalidSQLType
 - DataSetException class 4-138
- invalidStoreName
 - DataSetException class 4-138
- IO_ERROR
 - DataSetException class 4-125
- IOEXCEPTION
 - ResponseEvent class 4-244
- IOException
 - DataSetException class 4-138
- IS_CONTAINER
 - BasicBeanInfo class 8-10
- isAbort
 - ErrorResponse component 8-26
- isAccumulateResults
 - ProcedureDataSet class 5-36
 - Provider class 4-198
 - QueryDataSet class 5-61
- isAssignedNull
 - ReadRow class 4-217
 - Variant component 4-343
- isAsynchronousExecution
 - QueryDescriptor class 5-75
- isCancel
 - ResponseEvent class 4-247
- isCascadeDeletes
 - MasterLinkDescriptor class 4-173
- isCascadeUpdates
 - MasterLinkDescriptor class 4-173
- isCaseInsensitive
 - SortDescriptor class 4-267
 - StreamProperties class 2-48
- isCompatibleList
 - ReadRow class 4-217
- isComplete
 - ConnectionDescriptor class 5-7
 - ItemEditMask interface 6-34
- isConsistent
 - DataStore component 2-14
- isCopyProviderStreams
 - ProviderHelp class 4-203

- isCurrency
 - Column component 4-38
- isDescending
 - SortDescriptor class 4-269
 - StreamProperties class 2-50
- isDetailDataSetWithFetchAsNeeded
 - DataSet class 4-88
- isDisplayErrors
 - DataSet class 4-88
- isDoTransactions
 - SQLResolutionManager component 5-98
- isEditable
 - Column component 4-40
 - DataSet class 4-88
- isEditing
 - DataSet class 4-88
 - RowIterator class 4-254
- isEditingNewRow
 - DataSet class 4-89
 - RowIterator class 4-254
- isEmpty
 - DataSet class 4-89
- isEnabled
 - TxManager component 2-68
- isEnabledDelete
 - DataSet class 4-89
- isEnabledInsert
 - DataSet class 4-89
- isEnabledUpdate
 - DataSet class 4-89
- isEnforceIntegrity
 - PickListDescriptor class 4-196
- isExecuteOnOpen
 - QueryDescriptor class 5-75
- isFetchAsNeeded
 - MasterLinkDescriptor class 4-174
- isFixedPrecision
 - Column component 4-43
- isHidden
 - Column component 4-44
- isIgnore
 - ErrorResponse component 8-26
- isIgnoreAll
 - ResponseEvent class 4-247
- isLoadAsInserted
 - TextDataFile component 4-317
- isLoadOnOpen
 - DataFile class 4-77
 - TextDataFile component 4-317
- isModified
 - DataSet class 4-106
 - ReadRow class 4-217
- isNew
 - DataSet class 4-107
- isNull
 - ReadRow class 4-217
 - Variant component 4-346
- isOk
 - ResponseEvent class 4-247
- isOpen
 - Database component 5-18
 - DataSet class 4-90
 - DataStoreConnection component 2-22
- isPassword
 - ItemEditMaskStr class 6-40
- isPersist
 - Column component 4-46
- isPrimary
 - SortDescriptor class 4-268
 - StreamProperties class 2-49
- isPromptPassword
 - ConnectionDescriptor class 5-8
- isProviderPropertyChanged
 - ProviderHelp class 4-203
- isReadOnly
 - Column component 4-47
 - DataStore component 2-15
 - StorageDataSet component 4-281
- isReadOnlyTx
 - DataStoreConnection component 2-22
- isRecordStatus
 - TxManager component 2-69
- isReportConnect
 - DataStoreServer component 3-7
- isReportConnectError
 - DataStoreServer component 3-7
- isReportServerError
 - DataStoreServer component 3-7
- isRequired
 - Column component 4-48
- isResolvable
 - Column component 4-48
 - StorageDataSet component 4-282
- isResolve
 - ResolverResponse component 4-240
- isRetry
 - ErrorResponse component 8-26
- isRowId
 - Column component 4-48
- isSearchable
 - Column component 4-49
- isSkip
 - ResolverResponse component 4-240
- isSoftCommit
 - TxManager component 2-69
- isSortable
 - Column component 4-49
- isSortAsInserted
 - SortDescriptor class 4-268

- isTextual
 - Column component 4-50
- isTransparent
 - CustomPaintSite interface 4-71
- isUnassignedNull
 - ReadRow class 4-218
 - Variant component 4-347
- isUnique
 - SortDescriptor class 4-268
 - StreamProperties class 2-49
- isUpdatable
 - CustomAggOperator class 4-68
- isUseCaseSensitiveId
 - Database component 5-19
- isUseCaseSensitiveQuotedId
 - Database component 5-19
- isUseSchemaName
 - Database component 5-20
- isUseSetObjectForStreams
 - Database component 5-20
- isUseSetObjectForStrings
 - Database component 5-20
- isUseSpacePadding
 - Database component 5-20
- isUseStatementCaching
 - Database component 5-20
- isUseTableName
 - Database component 5-21
- isUseTransactions
 - Database component 5-21
- item editor classes
 - overview in DataExpress library 6-2
- item formatters
 - overview in DataExpress library 6-3
- ItemEditMask interface 6-32
- ItemEditMaskState component 6-36
 - methods 6-37
 - properties 6-37
- ItemEditMaskState constructor
 - ItemEditMaskState component 6-36, 6-37
- ItemEditMaskStr class 6-37
 - properties 6-39
- ItemEditMaskStr constructor
 - ItemEditMaskStr class 6-38
- itemEditor
 - Column component 4-44
- ItemFormatStr class 6-40
- ItemFormatStr constructor
 - ItemFormatStr class 6-44
- ItemFormatter class 6-47
- itemMargins
 - CustomPaintSite interface 4-71
- itemPainter
 - Column component 4-44

J

- Java Object
 - support for 4-33
- javaClass
 - Column component 4-44
- jb.io package
 - overview of classes 7-2
- jb.util package
 - overview of classes 8-2
- JDataStore library overview 1-1
- JDBC adapters 5-10
- JDBC connections
 - changing attributes 5-10, 5-12
 - debugging 5-15
- JDBC data sources
 - resolving changes to 5-52
- jdbcConnection
 - Database component 5-17

K

- keepLiterals
 - TextFormat component 6-67
- KEY_COLUMNS
 - UpdateMode interface 4-320
- keyCount
 - SortDescriptor class 4-269
- keys
 - ArrayResourceBundle class 8-5
 - SortDescriptor class 4-267

L

- LAST
 - Locate interface 4-167
- last
 - DataSet class 4-107
 - RowIterator class 4-260
- lastChar
 - FastStringBuffer component 8-43
- lastColumnVisited
 - DataSet class 4-90
- lastIndexOf
 - FastStringBuffer component 8-43
- LATE_SETTING
 - BasicBeanInfo class 8-10
- LEFT
 - Alignment class 6-6
- length
 - FastStringBuffer component 8-36, 8-43
- LESS_THAN_MIN
 - ValidationException class 4-329
- library overview 1-1
- LINK_COLUMNS_ERROR
 - DataSetException class 4-125

LINKFIELD_IN_USERPARAMETERS

- DataSetException class 4-126
- listener classes
 - CalcAggFieldsListener 4-24
 - CalcFieldsListener 4-26
 - CoerceToListener 4-30
 - ColumnChangeListener 4-55
 - ColumnPaintListener 4-60
 - ConnectionUpdateListener 5-12
 - DataChangeListener 4-76
 - EditListener 4-153
 - ExceptionListener 4-162
 - LoadListener 4-165
 - NavigationListener 4-183
 - OpenListener 4-184
 - ResolverListener 4-235
 - ResponseListener 4-249
 - StatusListener 4-275
- listener classes for datasets
 - overview in DataExpress library 4-6, 5-3
- listener registration 4-162
- listenerCount
 - EventMulticaster class 8-28
- listeners
 - database connections 5-12
 - EventMulticaster class 8-27
 - status messages 4-270
- literalAt
 - ItemEditMaskStr class 6-40
- load
 - DataFile class 4-78
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-70
 - StorageDataSet component 4-301
 - TableDataSet component 4-314
 - TextDataFile component 4-318
- Load interface 5-28
- loadAsInserted
 - TextDataFile component 4-317
- LoadCancel interface 4-162
- loadDataSet
 - DataSetData class 4-116
- LOADED
 - RowStatus interface 4-262
- loaders 4-163
- LoadEvent class 4-163
 - properties 4-164
 - variables 4-163
- LoadEvent constructor
 - LoadEvent class 4-164
- loading data
 - canceled operation 4-163
 - from result sets 5-72
 - implementing loader for 4-163
 - listener for 4-165

- multithreaded applications 4-165
- notifications for StorageDataSets 4-163
- LOADING_DATA
 - StatusEvent class 4-271
- LOADING_NOT_STARTED
 - DataSetException class 4-126
- LoadListener interface 4-165
- loadMetaData
 - DataFile class 4-78
 - TextDataFile component 4-318
- loadOnOpen
 - DataFile class 4-77
 - TextDataFile component 4-317
- loadOption
 - QueryDescriptor class 5-76
- loadRow
 - LoadRowListener interface 4-166
 - StorageDataSet component 4-295
- LoadRowListener interface 4-165
- locale
 - BinaryFormatter component 6-10
 - Column component 4-44
 - DataStore component 2-14
 - DataStoreConnection component 2-22
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-48
 - SimpleFormatter component 6-59
 - SortDescriptor class 4-267
 - StorageDataSet component 4-279
 - TextDataFile component 4-317
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- localeName
 - SortDescriptor class 4-267
- LocaleUtil class
 - properties 8-49
- localization variables 4-78
- locate
 - AggOperator class 4-23
 - DataSet class 4-107
 - MaxAggOperator class 4-177
 - MinAggOperator class 4-181
 - RowIterator class 4-260
- Locate interface 4-166
- LOCATE_MATCH_FOUND
 - StatusEvent class 4-271
- LOCATE_MATCH_NOT_FOUND
 - StatusEvent class 4-271
- LOCATE_NON_STRING
 - StatusEvent class 4-272
- LOCATE_STRING
 - StatusEvent class 4-272
- LOCATE_USE_ENTER
 - StatusEvent class 4-272

LOCATE_USE_MIXED_CASE

StatusEvent class 4-272

locating data 4-80

specifying options for 4-166

LOCK_TIMEOUT

TxException class 2-59

lockWaitTime

DataStoreConnection component 2-22

LOG_EXISTS

TxException class 2-59

logBlockSize

TxManager component 2-68

LOGDIR_MISMATCH

TxException class 2-59

LONG

Variant component 4-337

long

Variant component 4-345

LongFormatter component 6-50

LongFormatter constructor

LongFormatter component 6-50

LongType_S

Variant component 4-337

LOOKUP

CalcType interface 4-28

lookup

DataSet class 4-107

lookup columns

creating 4-27

defining pick lists for 4-193

lookup tables 4-184

lookupDisplayColumn

PickListDescriptor class 4-197

M

majorVersion

DataStoreDriver component 3-4

makeroom

FastStringBuffer component 8-43

mapping overrides 4-33

mark

FastBufferedInputStream class 7-13

markPendingStatus

ProviderHelp class 4-204

markSupported

FastBufferedInputStream class 7-13

masks B-1

date/time B-3

numeric B-1

setting default display 4-39

storing display formats 4-39

storing edit specifications 4-41

text B-4

MASTER_DETAIL_VIEW_ERROR

DataSetException class 4-126

MASTER_NAVIGATION_ERROR

DataSetException class 4-126

masterDataSet

MasterLinkDescriptor class 4-175

master-detail 5-58

master-detail relationships

fetching details 4-168

setting up persistent columns for 4-171

specifying 4-168

masterLink

DataSet class 4-90

DataSetView component 4-145

masterLinkColumns

MasterLinkDescriptor class 4-175

MasterLinkDescriptor class 4-168

methods 4-175

MasterLinkDescriptor constructor

MasterLinkDescriptor class 4-171, 4-172

masterNavigate

DataSet class 4-113

DataSetView component 4-151

ProcedureDataSet class 5-45

QueryDataSet class 5-70

StorageDataSet component 4-301

TableDataSet component 4-314

max

Column component 4-45

MAX_BIGDECIMAL_SCALE

DataStore component 2-11

MAX_CHECK_FREQUENCY

TxManager component 2-66

MAX_LOG_BLOCK_SIZE

TxManager component 2-66

MAX_OPEN_LOGS

TxManager component 2-66

MaxAggOperator class 4-176

properties 4-176

variables 4-176

maxDesignRows

StorageDataSet component 4-279

maximum values

aggregating 4-176

maxInline

Column component 4-45

maxLogSize

TxManager component 2-68

maxOpenLogs

TxManager component 2-68

maxResolveErrors

StorageDataSet component 4-280

maxRowLength

DataStore component 2-14

maxRowLocks

DataStore component 2-14

- maxRows
 - StorageDataSet component 4-280
- maxSortBuffer
 - DataStore component 2-15
- maxStatements
 - Database component 5-18
- MaxTypes
 - Variant component 4-338
- maxValue
 - Column component 4-45
- message
 - ResolveError class 5-95
 - ResponseEvent class 4-247
 - StatusEvent class 4-274
- messages
 - determining status type 4-270
 - getting content 4-275
- metaData
 - Database component 5-18
- metadata
 - suppressing generation of 5-57
 - update constants 4-177
- metaDataUpdate
 - StorageDataSet component 4-280
- MetaDataUpdate interface 4-177
- methodDescriptors
 - BasicBeanInfo class 8-12
- methodNames
 - BasicBeanInfo class 8-10
- methodParameters
 - BasicBeanInfo class 8-10
- MIDDLE
 - Alignment class 6-6
- min
 - Column component 4-45
- MIN_CHECK_FREQUENCY
 - TxManager component 2-66
- MIN_LOG_BLOCK_SIZE
 - TxManager component 2-66
- MIN_LOG_SIZE
 - TxManager component 2-66
- MIN_OPEN_LOGS
 - TxManager component 2-66
- MinAggOperator class 4-179
 - properties 4-180
 - variables 4-179
- minCacheSize
 - DataStore component 2-15
- minimum values
 - aggregating 4-179
- minorVersion
 - DataStoreDriver component 3-4
- minValue
 - Column component 4-46
- misc util classes
 - overview 8-3
- MISMATCH_PARAM_RESULT
 - DataSetException class 4-126
- MISMATCHED_PARAMETER_FORMAT
 - DataSetException class 4-126
- mismatchedParameterFormat
 - DataSetException class 4-138
- mismatchParamResult
 - DataSetException class 4-139
- MISSING_CHECKPOINT_FILE
 - TxException class 2-59
- MISSING_LOG_FILE
 - TxException class 2-59
- MISSING_MASTER_DATASET
 - DataSetException class 4-126
- MISSING_REPLACESTOREROW
 - DataSetException class 4-126
- MISSING_RESOLVER
 - DataSetException class 4-127
- missingMasterDataSet
 - DataSetException class 4-139
- mkUrlNotFound
 - DataSetException class 4-139
- mkUrlNotFoundInDesign
 - DataSetException class 4-139
- mobile users
 - creating data sources for 4-305
- modifying
 - EditListener interface 4-158
- move
 - ItemEditMask interface 6-35
- moveColumn
 - StorageDataSet component 4-296
- moveRow
 - DataSet class 4-108
- moving through data sets 4-181, 4-183
- multicaster classes
 - overview 8-3
- multiple column pick lists 4-193
- MULTIPLE_ROWS_AFFECTED
 - DataSetException class 4-127
- multipleRowsAffected
 - DataSetException class 4-139
- multiRowChange
 - DataChangeEvent class 4-75
- multi-table queries 5-58

N

- name conflicts 5-58
- NAME_NOT_UNIQUE
 - DataStoreException class 2-39
- namedAttributes
 - BasicBeanInfo class 8-10

- navigated
 - NavigationListener interface 4-183
- navigating 4-181, 4-183
- navigation
 - DataSet class 4-113
 - DataSetView component 4-151
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-70
 - StorageDataSet component 4-301
 - TableDataSet component 4-314
- NavigationEvent class 4-181
 - properties 4-182
 - variables 4-181
- NavigationEvent constructor
 - NavigationEvent class 4-181
- NavigationListener interface 4-183
- NEED_A_LOGDIR
 - TxException class 2-59
- NEED_LOCATE_START_OPTION
 - DataSetException class 4-127
- NEED_PROCEDUREPROVIDER
 - DataSetException class 4-127
- NEED_QUERYPROVIDER
 - DataSetException class 4-127
- NEED_STORAGEDATASET
 - DataSetException class 4-127
- needException
 - Diagnostic class 8-18
- needProcedureProvider
 - DataSetException class 4-139
- needQueryProvider
 - DataSetException class 4-139
- NEEDS_RECALC
 - DataSetException class 4-127
- needsAggDataSet
 - AggOperator class 4-23
- needsRecalc
 - DataSetException class 4-139
- needsRestructure
 - StorageDataSet component 4-281
- newColumn
 - AccessEvent class 4-15
- NEWER_VERSION
 - DataStoreException class 2-40
- newOrdinal
 - AccessEvent class 4-15
- NEXT
 - Locate interface 4-167
- next
 - DataSet class 4-108
 - ExceptionChain component 8-31
 - RowIterator class 4-260
- NEXT_FAST
 - Locate interface 4-167
- nextChar
 - FastStringBuffer component 8-43
- NO_CALC
 - CalcType interface 4-28
- NO_CALC_AGG_FIELDS
 - DataSetException class 4-127
- NO_CALC_FIELDS
 - DataSetException class 4-128
- NO_DATABASE_TO_RESOLVE
 - DataSetException class 4-128
- NO_NON_BLOB_COLUMNS
 - DataSetException class 4-128
- NO_PRIMARY_KEY
 - DataSetException class 4-128
- NO_PRIOR_ORIGINAL_ROW
 - DataSetException class 4-128
- NO_RESULT_SET
 - DataSetException class 4-128
- NO_ROWS_AFFECTED
 - DataSetException class 4-128
- NO_ROWS_TO_DELETE
 - ValidationException class 4-329
- NO_UPDATABLE_COLUMNS
 - DataSetException class 4-128
- NO_WHERE_CLAUSE
 - DataSetException class 4-129
- noDatabaseOnResolver
 - DataSetException class 4-139
- NON_EXISTENT_ROWID
 - DataSetException class 4-129
- NONE
 - MetaDataUpdate interface 4-178
 - ParameterType class 4-192
- nonExistentRowId
 - DataSetException class 4-140
- noResultSet
 - DataSetException class 4-140
- normalizeDelimiters
 - FastStringBuffer component 8-44
- noRowsAffected
 - DataSetException class 4-140
- NOT_A_CHAR
 - FastStringBuffer component 8-34
 - TextFormat component 6-66
- NOT_DATABASE_RESOLVER
 - DataSetException class 4-129
- NOT_NULL
 - Sort interface 4-263
- NOT_SELECT_QUERY
 - DataSetException class 4-129
- NOT_TRANSACTIONAL
 - TxException class 2-59
- NOT_UPDATEABLE
 - DataSetException class 4-129

- NOTACHAR
 - FastStringBuffer component 8-34
- notDatabaseResolver
 - DataSetException class 4-140
- notSelectQuery
 - DataSetException class 4-140
- notSortable
 - DataSetException class 4-140
- noUpdatableColumns
 - DataSetException class 4-140
- noWhereClause
 - DataSetException class 4-140
- null
 - Variant component 4-346
- NULL_COLUMN_NAME
 - DataSetException class 4-129
- NULL_TYPES
 - Variant component 4-338
- nullString
 - BooleanFormat component 6-16
- nullVariant
 - Variant component 4-338
- numeric masks B-1

O

- OBJECT
 - Variant component 4-338
- object
 - Variant component 4-346
- Object type
 - support for Object 4-33
- ObjectFormatter component 6-53
- ObjectFormatter constructor
 - ObjectFormatter component 6-53
- ObjectType_S
 - Variant component 4-338
- ODBC drivers 5-15
- off-line data access 4-305
- offset
 - FastStringBuffer component 8-36, 8-44
- OK
 - ResponseEvent class 4-245
- ok
 - ResponseEvent class 4-247, 4-249
- OLD_LOG
 - TxException class 2-60
- oldColumn
 - AccessEvent class 4-16
- OLDER_VERSION
 - DataStoreException class 2-40
- oldOrdinal
 - AccessEvent class 4-16
- ONEPASS_INPUT_STREAM
 - DataSetException class 4-129

- onePassInputStream
 - DataSetException class 4-140
- OPEN
 - AccessEvent class 4-11
- open
 - AggOperator class 4-23
 - Database component 5-18
 - DataSet class 4-90, 4-108, 4-113
 - DataSetView component 4-151
 - DataStoreConnection component 2-22, 2-29
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-71
 - StorageDataSet component 4-301
 - TableDataSet component 4-315
- OpenAdapter class 4-183
 - methods 4-184
 - properties 4-183
- openConnection
 - Database component 5-25
- openDetails
 - DataSet class 4-109
- openDirectory
 - DataStoreConnection component 2-29
- opened
 - OpenListener interface 4-185
- openFileStream
 - DataStoreConnection component 2-30
- OPENING
 - ConnectionUpdateEvent class 5-11
- opening
 - OpenListener interface 4-185
- opening data sets
 - adapter for JDBC objects 4-183
 - internal event 4-10
 - listener for 4-184
- OpenListener interface 4-184
 - adapter for 4-183
- openMode
 - DataStore component 2-15
- OPERATION_CANCELED
 - DataStoreException class 2-40
- options
 - SortDescriptor class 4-268
- ORACLE
 - SQLDialect interface 5-96
- Oracle 5-73
 - synonym 5-73
- Oracle drivers 5-33
- Oracle servers
 - synonyms and updatable queries 5-59
- Oracle stored procedures 5-33
 - running 5-30
- OracleProcedureProvider class 5-30
 - methods 5-31
 - properties 5-31

- ordinal
 - Column component 4-46
- ORIGINAL
 - RowStatus interface 4-262
- originalQueryString
 - QueryDataSet class 5-61
- ORIGINALS_STREAM
 - DataStore component 2-12
- OUT
 - ParameterType class 4-192
- out
 - Diagnostic class 8-15
- output 5-15
- output streams 4-114
- outputEnabled
 - Diagnostic class 8-15
- overview of classes
 - datastore package 2-2
 - datastore.jdbc package 3-2
 - dx.dataset package 4-4
 - dx.sql.dataset package 5-3
 - dx.text package 6-2
 - jb.io package 7-2
 - jb.util package 8-2
- overview of library 1-1

P

- Package (com.borland)
 - com.borland.datastore 2-1
 - com.borland.datastore.jdbc 3-1
 - com.borland.dx.dataset 4-1
 - com.borland.dx.sql.dataset 5-1
 - com.borland.dx.text 6-1
 - com.borland.jb.io 7-1
 - com.borland.jb.util 8-1
- packages
 - overview of datastore classes 2-2
 - overview of datastore.jdbc classes 3-2
 - overview of dx.dataset classes 4-4
 - overview of dx.sql.dataset classes 5-3
 - overview of dx.text classes 6-2
 - overview of jb.io classes 7-2
 - overview of jb.util classes 8-2
- packages overview 1-1
- painting
 - ColumnPaintListener interface 4-61
- PARAMETER_COUNT_MISMATCH
 - DataSetException class 4-129
- parameterCountMismatch
 - DataSetException class 4-141
- parameterized queries 4-185
 - for range comparisons 4-185
- parameterRow
 - ProcedureDataSet class 5-36
 - ProcedureProvider class 5-49
 - Provider class 4-198
 - QueryDataSet class 5-61
 - QueryDescriptor class 5-76
 - QueryProvider component 5-78
- ParameterRow component 4-185
- ParameterRow constructor
 - ParameterRow component 4-186
- parameterType
 - Column component 4-46
- ParameterType class 4-191
 - methods 4-192
 - properties 4-192
- paramString
 - DispatchableEvent class 8-24
- parse
 - BigDecimalFormatter class 6-9
 - BinaryFormatter component 6-12
 - BooleanFormat component 6-18
 - BooleanFormatter component 6-20
 - ByteFormatter class 6-22, 6-23
 - DateFormatter component 6-25
 - DoubleFormatter class 6-27
 - IntegerFormatter class 6-30
 - ItemFormatStr class 6-46
 - ItemFormatter class 6-50
 - LongFormatter component 6-52
 - ObjectFormatter component 6-55
 - ShortFormatter class 6-57
 - SimpleFormatter component 6-60, 6-61
 - StringFormatter component 6-64
 - TextFormat component 6-68
 - TimeFormatter component 6-71
 - TimestampFormatter component 6-73, 6-74
 - VariantFormatStr class 6-79
 - VariantFormatter class 6-83, 6-84
- parseBackSlash
 - FastStringBuffer component 8-44
- parseLiteral
 - FastStringBuffer component 8-44
- parseObject
 - BooleanFormat component 6-18
 - TextFormat component 6-69
- parsing B-1
- parsing user input 4-39
- PARTIAL
 - Locate interface 4-168
- PARTIAL_SEARCH_FOR_STRING
 - DataSetException class 4-130
- password
 - ConnectionDescriptor class 5-8
- pattern
 - BinaryFormatter component 6-11
 - ItemFormatStr class 6-45
 - ItemFormatter class 6-49
 - SimpleFormatter component 6-59

- VariantFormatStr class 6-76
- VariantFormatter class 6-81
- patterns B-1
- peekNextChar
 - FastStringBuffer component 8-45
- pending status 4-297
- PENDING_RESOLVED
 - RowStatus interface 4-262
- performance 5-57
 - fine-tuning 5-57
- persist
 - Column component 4-46
- persistent columns 4-32
 - setting up for master-detail relationships 4-171
- pick lists 4-193
- pickList
 - Column component 4-47
- pickListColumns
 - PickListDescriptor class 4-197
- pickListDataSet
 - PickListDescriptor class 4-197
- PickListDescriptor class 4-193
 - methods 4-197
- PickListDescriptor constructor
 - PickListDescriptor class 4-195
- pickListDisplayColumns
 - PickListDescriptor class 4-197
- placement-related util classes
 - overview in DataExpress library 6-4
- populating *See* providers
- port
 - DataStoreServer component 3-6
- post
 - DataSet class 4-109
 - RowIterator class 4-260
- POST_ROW
 - DataChangeEvent class 4-73
- postAllDataSets
 - DataSet class 4-109
 - StorageDataSet component 4-296
- posting notifications 4-76
- posting rows
 - validation error handling 4-320
- postRow
 - DataChangeListener interface 4-76
- PRECISION
 - MetaDataUpdate interface 4-178
- precision
 - Column component 4-47
- precondition
 - Diagnostic class 8-18
- preferredOrdinal
 - Column component 4-47
- prepare
 - ItemEditMask interface 6-35

- PRIMARY
 - Sort interface 4-263
- primary
 - SortDescriptor class 4-268
 - StreamProperties class 2-49
- print
 - Diagnostic class 8-18
- printDiagnosticStackTrace
 - ExceptionChain component 8-32
- printing conventions (documentation) 1-3
- println
 - Diagnostic class 8-19
- printlnc
 - Diagnostic class 8-19
- printStackTrace
 - DataSetException class 4-141
 - Diagnostic class 8-19
 - ExceptionChain component 8-32
- PRIOR
 - Locate interface 4-168
- prior
 - DataSet class 4-109
 - RowIterator class 4-260
- PRIOR_FAST
 - Locate interface 4-168
- priorChar
 - FastStringBuffer component 8-45
- procedure
 - ProcedureDataSet class 5-36
 - ProcedureProvider class 5-49
- PROCEDURE_FAILED
 - DataSetException class 4-130
- PROCEDURE_IN_PROCESS
 - DataSetException class 4-130
- ProcedureDataSet class 5-32
- ProcedureDataSet component
 - asynchronous fetching and 4-163
 - Oracle synonym 5-73
 - resolving changes 4-319
- ProcedureDescriptor class 5-45
 - methods 5-48
 - Oracle synonym 5-73
 - properties 5-48
- ProcedureDescriptor constructor
 - ProcedureDescriptor class 5-46, 5-47
- procedureFailed
 - DataSetException class 4-141
- procedureInProgress
 - DataSetException class 4-141
- ProcedureProvider class 5-48
- ProcedureResolver component 5-52
- ProcedureResolver constructor
 - ProcedureResolver component 5-52
- PROGRESS
 - StreamVerifier class 2-51

- promptPassword
 - ConnectionDescriptor class 5-8
- properties
 - ConnectionDescriptor class 5-8
- PROPERTY_CHANGE
 - AccessEvent class 4-11
- propertyDescriptorAttributes
 - BasicBeanInfo class 8-11
- propertyDescriptors
 - BasicBeanInfo class 8-11, 8-12
- provideData
 - Provider class 4-200
- provideMoreData
 - Provider class 4-200
 - StorageDataSet component 4-296
- provider
 - ProcedureDataSet class 5-36
 - QueryDataSet class 5-61
 - StorageDataSet component 4-281
- Provider class 4-198
- PROVIDER_FAILED
 - DataSetException class 4-130
- PROVIDER_OWNED
 - DataSetException class 4-130
- providerFailed
 - DataSetException class 4-141
- ProviderHelp class 4-201
 - properties 4-201
- providerOwned
 - DataSetException class 4-141
- providers 4-198
 - result sets 5-77
- providers for datasets
 - overview in DataExpress library 5-4
- providing
 - data type conversions A-3
- pseudo records 4-86

Q

- queries 5-71
 - asynchronous fetching and 4-163, 4-165
 - canceling load operations 4-163
 - editing detail queries 4-170
 - examples 5-71
 - executing 5-72
 - fetching detail data 4-170
 - multi-table 5-58
 - resolving changes 4-319
 - running 5-56
 - running against SQL views 5-59
 - updatable vs. read-only 5-56
- query
 - QueryDataSet class 5-62
 - QueryProvider component 5-78

- QUERY_FAILED
 - DataSetException class 4-130
- QUERY_IN_PROCESS
 - DataSetException class 4-130
- QueryDataSet class 5-56
- QueryDataSet component
 - canceling load operations 4-163
 - Oracle synonym 5-73
 - performance tuning 5-57
 - resolving changes 4-319
- QueryDescriptor class 5-71
 - Oracle synonym 5-73
- QueryDescriptor constructor
 - QueryDescriptor class 5-73, 5-74
- queryFailed
 - DataSetException class 4-141
- queryInProgress
 - DataSetException class 4-142
- QueryProvider component 5-77
- QueryProvider constructor
 - QueryProvider component 5-77
- QueryResolver component 5-81
- QueryResolver constructor
 - QueryResolver component 5-82
- queryString
 - ProcedureDataSet class 5-36
 - QueryDescriptor class 5-76

R

- range comparisons 4-185
- read
 - AsciiInputStream class 7-4
 - EncodedInputStream class 7-8
 - FastBufferedInputStream class 7-13
 - SimpleCharInputStream class 7-20
- READ_BLOCK_ERROR
 - DataStoreException class 2-40
- READ_ONLY
 - DataStoreException class 2-40
- READ_ONLY_COLUMN
 - ValidationException class 4-329
- READ_ONLY_DATASET
 - ValidationException class 4-329
- READ_ONLY_OPEN
 - ResponseEvent class 4-245
- READ_ONLY_STORE
 - DataSetException class 4-130
- reading entries 1-2
- readLine
 - SimpleCharInputStream class 7-20
- readObject
 - DataStoreConnection component 2-30
- readOnly
 - Column component 4-47
 - DataStore component 2-15

- StorageDataSet component 4-281
- read-only data sets
 - accessing 4-205
- read-only queries 5-56
- readOnlyStore
 - DataSetException class 4-142
- readOnlyTx
 - DataStoreConnection component 2-22
- readOnlyTxDelay
 - DataStoreConnection component 2-23
- readRow
 - RowIterator class 4-254
- ReadRow class 4-205
- readWriteRow
 - RowIterator class 4-254
- ReadWriteRow class 4-218
- reason
 - AccessEvent class 4-16
- recalc
 - StorageDataSet component 4-296
- recalculating data sets
 - listener for 4-26
- records *See* rows of data
- recordStatus
 - TxManager component 2-69
- RECOVERY_DENIED
 - TxException class 2-60
- reference entries 1-2
- refetchRow
 - DataSet class 4-109
 - QueryDataSet class 5-69
- refilter
 - DataSet class 4-110
- refresh
 - DataSet class 4-110
 - ProcedureDataSet class 5-43
 - QueryDataSet class 5-69
 - StorageDataSet component 4-296
- refreshing data
 - adapter for JDBC objects 4-59
- REFRESHROW_NOT_SUPPORTED
 - DataSetException class 4-131
- refreshSupported
 - DataSet class 4-110
 - ProcedureDataSet class 5-43
 - QueryDataSet class 5-69
 - StorageDataSet component 4-297
- remote methods
 - passing raw data to 4-114
- remove
 - EventMulticaster class 8-30
- removeChar
 - FastStringBuffer component 8-45
- removeCharAt
 - FastStringBuffer component 8-45
- removeChars
 - FastStringBuffer component 8-45
- removeCharsAt
 - FastStringBuffer component 8-46
- removeExceptionListener
 - DataSetException class 4-142
- removeLoadRowListener
 - StorageDataSet component 4-297
- removeServerStatusListener
 - DataStoreServer component 3-9
- removeTraceCategory
 - Diagnostic class 8-19
- renameStream
 - DataStoreConnection component 2-30
- REOPEN_FAILURE
 - DataSetException class 4-131
- replaceCharacter
 - TextFormat component 6-67
- replaceCharAt
 - FastStringBuffer component 8-46
- reportConnect
 - DataStoreServer component 3-7
- reportConnectError
 - DataStoreServer component 3-7
- reportServerError
 - DataStoreServer component 3-7
- required
 - Column component 4-48
- requiredColumnsCheck
 - ReadWriteRow class 4-222
- reset
 - CustomPaintSite interface 4-71
 - FastBufferedInputStream class 7-14
 - StorageDataSet component 4-297
- resetInBounds
 - DataSet class 4-110
- resetPendingStatus
 - Database component 5-25
 - DataSet class 4-110, 4-111
 - StorageDataSet component 4-297
- ResolutionException class 5-85
 - methods 5-91
- ResolutionException constructor
 - ResolutionException class 5-89, 5-90
- resolvable
 - Column component 4-48
 - StorageDataSet component 4-282
- resolve
 - ResolverResponse component 4-240, 4-241
- RESOLVE_FAILED
 - DataSetException class 4-131
 - ResolutionException class 5-89
- RESOLVE_IN_PROGRESS
 - DataSetException class 4-131

- RESOLVE_PARTIAL
 - ResolutionException class 5-89
- resolveData
 - Resolver class 4-233
 - SQLResolver class 5-102
- ResolveError class 5-93
 - methods 5-96
 - properties 5-95
- resolveFailed
 - DataSetException class 4-142
- resolveOrder
 - StorageDataSet component 4-282
- resolver
 - ProcedureResolver component 5-55
 - QueryResolver component 5-85
 - SQLResolver class 5-102
 - StorageDataSet component 4-282
- Resolver class 4-232
 - properties 4-232
- ResolverAdapter class 4-233
 - methods 4-234
 - properties 4-234
- ResolverListener interface 4-235
 - adapter for 4-233
- resolverQueryTimeout
 - QueryResolver component 5-83
- ResolverResponse component 4-239
 - variables 4-240
- ResolverResponse constructor
 - ResolverResponse component 4-240
- resolvers 4-232
 - error handling 5-85
 - for result sets 5-81
 - initializing 5-28
 - JDBC data sources 5-52
 - providing notifications for 4-233, 4-235
 - SQL data changes 5-100
 - SQL transactions 5-97
 - testing if needed 4-239
- resolvers for datasets
 - overview in DataExpress library 4-8, 5-4
- resolving changes
 - queries 4-319
 - stored procedures 4-319
- resourceable SQL 5-72
- ResourceBundles 5-72
- response
 - DataStore component 2-19
 - ErrorResponse component 8-25, 8-26
 - ResolveError class 5-95
 - ResponseEvent class 4-247
 - ResponseListener interface 4-249
- ResponseAdapter class 4-241
 - methods 4-242
 - properties 4-242
- ResponseEvent class 4-242
- ResponseEvent constructor
 - ResponseEvent class 4-246
- ResponseListener interface 4-249
 - adapter for 4-241
- responses (user-requested) 4-241
 - collecting 4-242
 - listener for 4-249
- restructure
 - StorageDataSet component 4-298
- RESTRUCTURE_DATA_LOSS
 - DataStoreException class 2-40
- RESTRUCTURE_IN_PROGRESS
 - DataSetException class 4-131
- RESTRUCTURE_PARSE_ERROR
 - DataStoreException class 2-41
- RESTRUCTURE_PRECISION_LOSS
 - DataStoreException class 2-41
- RESTRUCTURING
 - StatusEvent class 4-272
- restructuring data sets
 - internal event 4-10
- RESULT
 - ParameterType class 4-192
- result sets 5-33, 5-56
 - loading 5-72
 - populating 5-77
 - resolving changes to 5-81
- resultColumn
 - AggOperator class 4-21
- resultSetToDataSet
 - Database component 5-25, 5-26
- resultValue
 - AggOperator class 4-21
- RETRY
 - ErrorResponse component 8-25
- retry
 - ErrorResponse component 8-26, 8-27
- RETURN
 - ParameterType class 4-192
- RIGHT
 - Alignment class 6-6
- rollback
 - Database component 5-26
 - DataStoreConnection component 2-31
- row
 - DataSet class 4-90
 - ResolveError class 5-95
 - RowIterator class 4-254
- ROW_ADDED
 - DataChangeEvent class 4-73
- ROW_CHANGE_POSTED
 - DataChangeEvent class 4-73
- ROW_CHANGED
 - DataChangeEvent class 4-73

- ROW_DELETED
 - DataChangeEvent class 4-74
- ROW_LOCK_TIMEOUT
 - TxException class 2-60
- ROW_NOT_FOUND
 - DataStoreException class 2-41
- ROW_TOO_WIDE
 - DataStoreException class 2-41
- rowAffected
 - DataChangeEvent class 4-74
- rowCount
 - DataSet class 4-91
- rowFilter
 - DataSet class 4-113
 - DataSetView component 4-151
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-71
 - StorageDataSet component 4-301
 - TableDataSet component 4-315
- rowFilterListener
 - DataSet class 4-91
- RowFilterListener interface 4-249
- RowFilterResponse class 4-250
 - properties 4-251
- ROWID
 - MetaDataUpdate interface 4-178
- rowId
 - Column component 4-48
- RowIterator class 4-252
- row-related classes
 - overview in DataExpress library 4-9
- rows of data 4-80
 - current row listener 4-183
 - posting notifications 4-76
 - providing read access 4-205
 - providing write access 4-218
 - temporary storage for 4-86
- RowStatus interface 4-261
- runtimeMetaData
 - Database component 5-18

S

- sample applications
 - calculations 4-27
 - custom provider/resolver for DataSetData objects 4-114
 - parameterized queries 4-185
 - providers 4-198
 - resolvers 4-232
 - stored procedures 5-32
- save
 - DataFile class 4-78
 - DataStore component 2-18
 - TextDataFile component 4-319
 - saveChanges
 - Database component 5-26, 5-27
 - DataSet class 4-111
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-69
 - StorageDataSet component 4-298
 - saveChangesSupported
 - DataSet class 4-111
 - ProcedureDataSet class 5-44
 - QueryDataSet class 5-69
 - StorageDataSet component 4-298
 - saveInterval
 - DataStore component 2-15
 - saveMode
 - DataStore component 2-16
 - saving 4-306
 - TableDataSet to JDBC 4-306
- SCALE
 - MetaDataUpdate interface 4-179
- scale
 - Column component 4-48
 - VariantFormatStr class 6-76
 - VariantFormmatter class 6-81
- schemaName
 - Column component 4-49
 - DataSet class 4-91
 - StorageDataSet component 4-283
- SEARCHABLE
 - MetaDataUpdate interface 4-179
- searchable
 - Column component 4-49
- searching data 4-80
 - specifying options for 4-166
- SearchPath class 8-50
 - properties 8-50
- SearchPath constructor
 - SearchPath class 8-50
- SECOND_INDEX_STREAM
 - DataStore component 2-12
- self-joins 4-170
- separator
 - TextDataFile component 4-317
- SERVER_ERROR
 - ServerStatus interface 3-12
- serverColumnName
 - Column component 4-49
- serverConnection
 - ServerStatusEvent class 3-13
- ServerConnection class 3-9
 - variables 3-9
- serverConnections
 - DataStoreServer component 3-7
- ServerStatus interface 3-11
- ServerStatusEvent class 3-12
 - variables 3-12

- ServerStatusListener interface 3-13
- serverStatusMessage
 - ServerStatusListener interface 3-14
- SET_CALCULATED_FAILURE
 - DataSetException class 4-131
- setAccumulateResults
 - ProcedureDataSet class 5-36
 - QueryDataSet class 5-61
- setAgg
 - Column component 4-36
- setAlignment
 - Column component 4-36
 - CustomPaintSite interface 4-70
- setAllRowIds
 - StorageDataSet component 4-278
- setALogDir
 - TxManager component 2-67
- setArrayLength
 - Variant component 4-341
- setAsDate
 - Variant component 4-342
- setAsObject
 - Variant component 4-350
- setAssignedNull
 - ReadWriteRow class 4-219
 - Variant component 4-350
- setAsTime
 - Variant component 4-343
- setAsTimestamp
 - Variant component 4-343
- setAsVariant
 - Variant component 4-343
- setAsynchronousExecution
 - QueryDescriptor class 5-75
- setAutoCommit
 - Database component 5-17
- setBackground
 - Column component 4-36
 - CustomPaintSite interface 4-70
- setBigDecimal
 - ReadWriteRow class 4-223
 - Variant component 4-344
- setBinaryStream
 - Variant component 4-344
- setBlockSize
 - DataStore component 2-14
- setBLogDir
 - TxManager component 2-67
- setBoolean
 - ReadWriteRow class 4-223
 - Variant component 4-344
- setByte
 - ReadWriteRow class 4-224
 - Variant component 4-344
- setByteArray
 - ReadWriteRow class 4-224
 - Variant component 4-350
- setCalcType
 - Column component 4-36
- setCaption
 - Column component 4-37
- setCharAt
 - FastStringBuffer component 8-46
- setCheckFrequency
 - TxManager component 2-68
- setColumnName
 - Column component 4-37
 - ColumnAware interface 4-54
- setColumns
 - ParameterRow component 4-186
 - StorageDataSet component 4-278
- setConnection
 - Database component 5-17
- setConnectionURL
 - ConnectionDescriptor class 5-7
- setCurrency
 - Column component 4-38
- setDatabase
 - ProcedureResolver component 5-53
 - QueryResolver component 5-83
 - SQLResolutionManager component 5-98
 - SQLResolver class 5-100
- setDataFile
 - StorageDataSet component 4-278
- setDataSet
 - DataSetAware interface 4-114
- setDataType
 - Column component 4-38
- setDate
 - ReadWriteRow class 4-224, 4-225
 - Variant component 4-344
- setDefault
 - Column component 4-39
- setDefaultValue
 - Column component 4-39
- setDefaultValues
 - DataSet class 4-88, 4-111
 - ReadWriteRow class 4-225
- setDeleteProcedure
 - ProcedureResolver component 5-53
- setDelimiter
 - TextDataFile component 4-316
- setDisplayErrors
 - DataSet class 4-88
- setDisplayMask
 - Column component 4-39, 4-52
- setDisplayVariant
 - DataSet class 4-111

- setDoTransactions
 - SQLResolutionManager component 5-98
- setDouble
 - ReadWriteRow class 4-225, 4-226
 - Variant component 4-345
- setDriver
 - ConnectionDescriptor class 5-7
- setEditable
 - Column component 4-40
 - DataSet class 4-88
- setEditMask
 - Column component 4-41, 4-52
- setEditMasker
 - Column component 4-41
- setEnabled
 - TxManager component 2-68
- setEnabledDelete
 - DataSet class 4-89
- setEnabledInsert
 - DataSet class 4-89
- setEnabledUpdate
 - DataSet class 4-89
- setEncoding
 - TextDataFile component 4-316
- setExecuteOnOpen
 - QueryDescriptor class 5-75
- setExportDisplayMask
 - Column component 4-42, 4-52
- setExportFormatter
 - Column component 4-43
- setFileFormat
 - TextDataFile component 4-317
- setFileName
 - DataStoreConnection component 2-21
 - TextDataFile component 4-317
- setFillCharacter
 - TextFormat component 6-66
- setFixedPrecision
 - Column component 4-43
- setFloat
 - ReadWriteRow class 4-226
 - Variant component 4-345
- setFont
 - Column component 4-43
 - CustomPaintSite interface 4-70
- setForeground
 - Column component 4-43
 - CustomPaintSite interface 4-71
- setFormatter
 - Column component 4-43
- setFromDouble
 - VariantFormatter class 6-84
- setFromInt
 - VariantFormatter class 6-85
- setFromString
 - Variant component 4-350
- setHidden
 - Column component 4-44
- setIdentifierQuoteChar
 - Database component 5-17
- setInputStream
 - ReadWriteRow class 4-226, 4-227
 - Variant component 4-345
- setInsertProcedure
 - ProcedureResolver component 5-53
- setInt
 - ReadWriteRow class 4-227
 - Variant component 4-345
- setItemEditor
 - Column component 4-44
- setItemMargins
 - CustomPaintSite interface 4-71
- setItemPainter
 - Column component 4-44
- setJavaClass
 - Column component 4-44
- setJdbcConnection
 - Database component 5-17
- setKeepLiterals
 - TextFormat component 6-67
- setLastColumnVisited
 - DataSet class 4-90
- setLength
 - FastStringBuffer component 8-36
- setLoadAsInserted
 - TextDataFile component 4-317
- setLoadOnOpen
 - TextDataFile component 4-317
- setLoadOption
 - QueryDescriptor class 5-76
- setLocale
 - Column component 4-44
 - DataStore component 2-14
 - StorageDataSet component 4-279
 - TextDataFile component 4-317
- setLockWaitTime
 - DataStoreConnection component 2-22
- setLogBlockSize
 - TxManager component 2-68
- setLogStream
 - Diagnostic class 8-19
- setLong
 - ReadWriteRow class 4-227, 4-228
 - Variant component 4-345
- setMasterLink
 - DataSet class 4-90
 - DataSetView component 4-145
- setMax
 - Column component 4-45

- setMaxDesignRows
 - StorageDataSet component 4-279
- setMaxInline
 - Column component 4-45
- setMaxLogSize
 - TxManager component 2-68
- setMaxOpenLogs
 - TxManager component 2-68
- setMaxResolveErrors
 - StorageDataSet component 4-280
- setMaxRowLocks
 - DataStore component 2-14
- setMaxRows
 - StorageDataSet component 4-280
- setMaxSortBuffer
 - DataStore component 2-15
- setMaxValue
 - Column component 4-45
- setMessage
 - StatusEvent class 4-274
- setMetaDataMissing
 - ProviderHelp class 4-204
- setMetaDataUpdate
 - StorageDataSet component 4-280
- setMin
 - Column component 4-45
- setMinCacheSize
 - DataStore component 2-15
- setMinValue
 - Column component 4-46
- setNull
 - Variant component 4-346
- setObject
 - ReadWriteRow class 4-228
 - Variant component 4-346
- setOffset
 - FastStringBuffer component 8-36
- setOpenMode
 - DataStore component 2-15
- setParameterRow
 - ProcedureProvider class 5-49
 - Provider class 4-198
 - QueryProvider component 5-78
- setParameterType
 - Column component 4-46
- setPassword
 - ConnectionDescriptor class 5-8
- setPattern
 - BinaryFormatter component 6-13
 - ItemFormatStr class 6-47
 - ItemFormatter class 6-50
 - SimpleFormatter component 6-61
 - VariantFormatStr class 6-79
 - VariantFormatter class 6-85
- setPersist
 - Column component 4-46
- setPickList
 - Column component 4-47
- setPort
 - DataStoreServer component 3-6
- setPrecision
 - Column component 4-47
- setPreferredOrdinal
 - Column component 4-47
- setProcedure
 - ProcedureDataSet class 5-36
 - ProcedureProvider class 5-49
- setPromptPassword
 - ConnectionDescriptor class 5-8
- setProperties
 - ConnectionDescriptor class 5-8
- setProvider
 - ProcedureDataSet class 5-36
 - QueryDataSet class 5-61
 - StorageDataSet component 4-281
- setProviderPropertyChanged
 - ProviderHelp class 4-204
- setQuery
 - QueryDataSet class 5-62
 - QueryProvider component 5-78
- setReadOnly
 - Column component 4-47
 - DataStore component 2-15
 - StorageDataSet component 4-281
- setReadOnlyTx
 - DataStoreConnection component 2-22
- setReadOnlyTxDelay
 - DataStoreConnection component 2-23
- setRecordStatus
 - TxManager component 2-69
- setReplaceCharacter
 - TextFormat component 6-67
- setReportConnect
 - DataStoreServer component 3-7
- setReportConnectError
 - DataStoreServer component 3-7
- setReportServerError
 - DataStoreServer component 3-7
- setRequired
 - Column component 4-48
- setResolvable
 - Column component 4-48
 - StorageDataSet component 4-282
- setResolveOrder
 - StorageDataSet component 4-282
- setResolver
 - StorageDataSet component 4-282
- setResolverQueryTimeout
 - QueryResolver component 5-83

- setRowId
 - Column component 4-48
 - StorageDataSet component 4-298
- setRuntimeMetaData
 - Database component 5-18
- setSaveInterval
 - DataStore component 2-15
- setSaveMode
 - DataStore component 2-16
- setScale
 - Column component 4-48
- setSchemaName
 - Column component 4-49
 - StorageDataSet component 4-283
- setSearchable
 - Column component 4-49
- setSeparator
 - TextDataFile component 4-317
- setServerColumnName
 - Column component 4-49
- setShort
 - ReadWriteRow class 4-228, 4-229
 - Variant component 4-346
- setSoftCommit
 - TxManager component 2-69
- setSort
 - DataSet class 4-91
- setSortPrecision
 - Column component 4-49
- setSpecialObject
 - BinaryFormatter component 6-13
 - ItemFormatStr class 6-47
 - SimpleFormatter component 6-61
 - VariantFormatStr class 6-79
 - VariantFormatter class 6-85
- setSQLDialect
 - Database component 5-18
- setSqlType
 - Column component 4-50
- setStorageDataSet
 - DataSetView component 4-145
- setStore
 - StorageDataSet component 4-283
- setStoreClassFactory
 - StorageDataSet component 4-283
- setStoreName
 - StorageDataSet component 4-283
- setString
 - ReadWriteRow class 4-229
 - Variant component 4-346
- setTableName
 - Column component 4-50
 - StorageDataSet component 4-284
- setTempDir
 - DataStoreServer component 3-7
- setTempDirName
 - DataStore component 2-16
- setTime
 - ReadWriteRow class 4-229, 4-230
 - Variant component 4-347
- setTimestamp
 - ReadWriteRow class 4-230, 4-231
 - Variant component 4-347, 4-351
- setting 4-17
 - aggregation type 4-17
 - default display masks 4-39
 - master-detail links 4-168
 - values *See* values
- setTraceLevel
 - Diagnostic class 8-20
- setTransactionIsolation
 - Database component 5-18
- setTransactionSupport
 - SQLResolutionManager component 5-98
- setTxIsolation
 - DataStoreConnection component 2-23
- setTxManager
 - DataStore component 2-16
- setType
 - Variant component 4-346
- setUnassignedNull
 - ReadWriteRow class 4-219
 - Variant component 4-351
- setUpdateMode
 - QueryResolver component 5-83
- setUpdateProcedure
 - ProcedureResolver component 5-53
- setUseCaseSensitiveId
 - Database component 5-19
- setUseCaseSensitiveQuotedId
 - Database component 5-19
- setUserName
 - ConnectionDescriptor class 5-8
 - DataStoreConnection component 2-23
- setUseSchemaName
 - Database component 5-20
- setUseSetObjectForStreams
 - Database component 5-20
- setUseSetObjectForStrings
 - Database component 5-20
- setUseSpacePadding
 - Database component 5-20
- setUseStatementCaching
 - Database component 5-20
- setUseTableName
 - Database component 5-21
- setUseTransactions
 - Database component 5-21
- setVariant
 - ReadWriteRow class 4-231, 4-232

- Variant component 4-347
- setVisible
 - Column component 4-50
- setWidth
 - Column component 4-50
- shiftLeft
 - ItemEditMaskStr class 6-40
- SHORT
 - Variant component 4-338
- short
 - Variant component 4-346
- ShortFormatter class 6-55
 - properties 6-56
 - variables 6-55
- ShortFormatter constructor
 - ShortFormatter class 6-56
- ShortType_S
 - Variant component 4-338
- shutdown
 - DataStore component 2-18
 - DataStoreServer component 3-9
- SILENT
 - StreamVerifier class 2-51
- SILENT_VERBOSE
 - StreamVerifier class 2-51
- SILENT_VERBOSE_EXCEPTION
 - StreamVerifier class 2-51
- SimpleCharInputStream class 7-19
 - properties 7-19
- SimpleCharOutputStream class 7-21
 - properties 7-21
- SimpleFormatter component 6-58
- SimpleFormatter constructor
 - SimpleFormatter component 6-58
- siteComponent
 - CustomPaintSite interface 4-71
- skip
 - FastBufferedInputStream class 7-14
 - ResolverResponse component 4-240, 4-241
- softCommit
 - TxManager component 2-69
- sort
 - DataSet class 4-91
- Sort interface 4-262
- sort order 4-263
 - applying to alternate views 4-144
 - getting locale 4-267
 - specifying locale 4-267
- SORT_AS_INSERTED
 - Sort interface 4-263
- sortable
 - Column component 4-49
- sortAsInserted
 - SortDescriptor class 4-268
- SortDescriptor class 4-263
- SortDescriptor constructor
 - SortDescriptor class 4-264, 4-265, 4-266
- SORTING
 - StatusEvent class 4-272
- sorting data 4-144, 4-263
 - getting locale 4-267
 - specifying locale 4-267
- sortPrecision
 - Column component 4-49
- sortTable
 - DataStoreConnection component 2-31
- sourceToText
 - FastStringBuffer component 8-46
- SQL +servers 5-13
- SQL connections 5-14
 - setting properties for 5-14
 - storing 5-5
- SQL data sources 5-32
 - accessing 5-32
- SQL databases
 - See also* SQL tables
- SQL queries 5-71
 - examples 5-71
 - multi-table 5-58
 - running 5-56
 - updatable vs. read-only 5-56
- SQL statements
 - executing 5-72
 - specifying same column multiple times 4-185
- SQL stored procedures 5-32, 5-72
 - calling 5-72
 - calling for JDBC data sources 5-52
 - running 5-48
 - storing properties 5-45
- SQL tables 5-13
 - changing data 5-100
 - column aliases 5-58
- SQL views 5-59
- SQL_ERROR
 - DataSetException class 4-131
- SQLDialect
 - Database component 5-18
- SQLDialect interface 5-96
- SQLException
 - DataSetException class 4-142
- SQLResolutionManager component 5-97
- SQLResolutionManager constructor
 - SQLResolutionManager component 5-98
- SQLResolver class 5-100
- sqlType
 - Column component 4-50
- START_MASK
 - Locate interface 4-168

- startEdit
 - DataSet class 4-112
- startEditCheck
 - DataSet class 4-112
- startLoading
 - StorageDataSet component 4-299, 4-300
- startResolution
 - ProviderHelp class 4-204
- status
 - DataSet class 4-91, 4-113
 - DataSetView component 4-151
 - ProcedureDataSet class 5-45
 - QueryDataSet class 5-71
 - StorageDataSet component 4-301
 - TableDataSet component 4-315
- status bar
 - customizing messages 4-275
 - determining message type 4-270
 - disabling messages 4-275
- status settings 4-261
- statusCode
 - ServerStatusEvent class 3-13
- StatusEvent class 4-270
- StatusEvent constructor
 - StatusEvent class 4-273
- StatusListener interface 4-274
- statusMessage
 - DataSet class 4-112
 - StatusListener interface 4-275
- storageDataSet
 - DataSet class 4-92
 - DataSetView component 4-145
- StorageDataSet component 4-275
 - creating from non-SQL sources 4-305
 - load notifications 4-163
 - populating 5-30, 5-48
- StorageDataSet constructor
 - StorageDataSet component 4-276
- store
 - StorageDataSet component 4-283
- STORE_NAME_NOT_SET
 - DataStoreException class 2-41
- STORE_OPERATION_UNSUPPORTED
 - DataStoreException class 2-42
- STORE_VERSION
 - DataStore component 2-12
- storeClassFactory
 - StorageDataSet component 4-283
- StoreClassFactory interface 4-302
- stored functions 5-33
- stored procedures 5-32
 - asynchronous fetching and 4-163, 4-165
 - calling 5-72
 - calling for JDBC data sources 5-52
 - example for running 5-32
 - executing 5-72
 - resolving changes 4-319
 - running 5-48
 - storing properties 5-45
- storeInternals
 - DataStoreConnection component 2-23
- storeName
 - StorageDataSet component 4-283
- storesLowerCaseIdentifiers
 - Database component 5-27
- storesUpperCaseIdentifiers
 - Database component 5-27
- storing data 4-78
 - for specific column 4-80
- STREAM_CLOSED
 - DataStoreException class 2-42
- STREAM_NOT_FOUND
 - DataStoreException class 2-42
- STREAM_OPEN_TWICE
 - DataStoreException class 2-42
- StreamProperties class 2-48
- StreamVerifier class 2-50
 - properties 2-52
- STRING
 - Variant component 4-338
- string
 - Variant component 4-346
- String-based patterns B-1
- StringFormatter component 6-62
- StringFormatter constructor
 - StringFormatter component 6-62
- stringFromChar
 - FastStringBuffer component 8-46
- StringType_S
 - Variant component 4-339
- STRUCTURE_CHANGE
 - AccessEvent class 4-12
- substring
 - FastStringBuffer component 8-47
- subtract
 - Variant component 4-351
- SumAggOperator class 4-303
 - properties 4-303
 - variables 4-303
- summary aggregations 4-303
- support options 1-3
- Sybase stored procedures 5-34
 - calling 5-52
- sync
 - DataStore component 2-19

T

- TABLE_FILE_STREAM
 - DataStore component 2-12

- TABLE_STREAM
 - DataStore component 2-12
- TableDataSet component 4-305
 - methods 4-307
 - properties 4-306
 - saving to a JDBC data source 4-306
- TableDataSet constructor
 - TableDataSet component 4-306
- tableExists
 - DataStoreConnection component 2-31
- TABLERNAME
 - MetaDataUpdate interface 4-179
- tableName
 - Column component 4-50
 - DataSet class 4-92
 - StorageDataSet component 4-284
- tables 4-305
- technical support 1-3
- tempDir
 - DataStoreServer component 3-7
- tempDirName
 - DataStore component 2-16
- terminating database connections 5-10, 5-12
- text files 4-315
 - canceling load operations 4-163
 - importing from 4-305
- text masks B-4
- TextDataFile component 4-315
- TextDataFile constructor
 - TextDataFile component 4-316
- TextFormat component 6-64
- TextFormat constructor
 - TextFormat component 6-66
- textToSource
 - FastStringBuffer component 8-47
- textual
 - Column component 4-50
- three-tier applications 4-114
- throwException
 - DataSetException class 4-142
- throwExceptionChain
 - DataSetException class 4-142
- TIME
 - Variant component 4-339
- time
 - Variant component 4-347
- TimeFormatter component 6-69
- TimeFormatter constructor
 - TimeFormatter component 6-69
- TIMESTAMP
 - Variant component 4-339
- timestamp
 - Variant component 4-347
- TimestampFormatter component 6-71
- TimestampFormatter constructor
 - TimestampFormatter component 6-72
- TimestampType_S
 - Variant component 4-339
- TimeType_S
 - Variant component 4-339
- toggleViewOrder
 - DataSet class 4-112
- TOO_MANY_ERRORS
 - DataStoreException class 2-42
- TOO_MANY_OPEN_FILES
 - TxException class 2-60
- TOP
 - Alignment class 6-6
- toPattern
 - BooleanFormat component 6-18
 - TextFormat component 6-69
- toString
 - Column component 4-53
 - ConnectionDescriptor class 5-9
 - DataChangeEvent class 4-75
 - DispatchableEvent class 8-24
 - FastStringBuffer component 8-47
 - QueryDescriptor class 5-77
 - ReadRow class 4-218
 - SearchPath class 8-51
 - SortDescriptor class 4-269
 - Variant component 4-351
- trace
 - Diagnostic class 8-20
- TRANSACTION_ISOLATION_LEVEL_NOT_SUPPORTED
 - DataSetException class 4-131
- transactionIsolation
 - Database component 5-18
- transactionIsolationLevelNotSupported
 - DataSetException class 4-142
- transactions 5-14
 - resolving changes 5-97
- transactionStarted
 - DataStoreConnection component 2-31
- transactionSupport
 - SQLResolutionManager component 5-98
- transparent
 - CustomPaintSite interface 4-71
- TriState classes
 - overview 8-3
- TriStateProperty interface 8-51
- troubleshooting 5-15, 5-61
 - JDBC connections 5-15
 - providing 5-61
- TRUE
 - TriStateProperty interface 8-52

- trueString
 - BooleanFormat component 6-16
- TX_MANAGER_INUSE
 - TxException class 2-60
- TX_NOT_ALLOWED
 - TxException class 2-60
- TX_REQUIRED
 - TxException class 2-61
- TxException class 2-54
 - methods 2-61
 - properties 2-61
- txIsolation
 - DataStoreConnection component 2-23
- txManager
 - DataStore component 2-16
- TxManager component 2-63
 - methods 2-69
- TxManager constructor
 - TxManager component 2-67
- type
 - IntegerFormatter class 6-28
 - Variant component 4-347
- type coercions A-3, A-4
 - caution for 4-33
 - from source to column 4-30
- type mappings
 - overriding 4-33
- TYPE_CHANGE_DATA_LOSS
 - ResponseEvent class 4-245
- TYPE_CHANGE_PARSE_ERROR
 - ResponseEvent class 4-245
- TYPE_CHANGE_PARSE_ERROR_TOTAL
 - ResponseEvent class 4-245
- TYPE_CHANGE_PRECISION_LOSS
 - ResponseEvent class 4-246
- typedId
 - Variant component 4-351
- typeName
 - StreamVerifier class 2-52
 - Variant component 4-352
- typeOf
 - Variant component 4-352
- types
 - caution for assigning 4-33
- typography 1-3

U

- UI components
 - rebinding 4-144
- UNASSIGNED
 - UpdateMode interface 4-320
- UNASSIGNED_NULL
 - Variant component 4-339
- unassignedNull
 - ReadWriteRow class 4-219

- Variant component 4-347
- UnassignedNull_S
 - Variant component 4-339
- unbind
 - RowIterator class 4-260
- UNCACHED
 - Load interface 5-30
- UNDEFINED
 - Alignment class 6-6
- undeleteStream
 - DataStoreConnection component 2-31
- UNEXPECTED_END_OF_QUERY
 - DataSetException class 4-131
- unexpectedEndOfQuery
 - DataSetException class 4-143
- UNIQUE
 - Sort interface 4-263
- unique
 - SortDescriptor class 4-268
 - StreamProperties class 2-49
- UNKNOWN
 - AccessEvent class 4-12
 - SQLDialect interface 5-96
- UNKNOWN_COLUMN_NAME
 - DataSetException class 4-132
- UNKNOWN_DETAIL_NAME
 - DataSetException class 4-132
- UNKNOWN_PARAM_NAME
 - DataSetException class 4-132
- unknownColumnName
 - DataSetException class 4-143
- unknownDetailName
 - DataSetException class 4-143
- unknownParamName
 - DataSetException class 4-143
- UnknownType_S
 - Variant component 4-339
- unread
 - AsciiInputStream class 7-4
 - EncodedInputStream class 7-8
 - FastBufferedInputStream class 7-14
 - SimpleCharInputStream class 7-21
- UNRECOGNIZED_DATA_TYPE
 - DataSetException class 4-132
- unrecognizedDataType
 - DataSetException class 4-143
- UNSPECIFIED
 - AccessEvent class 4-12
- updatable
 - CustomAggOperator class 4-68
- updatable queries 5-56
 - creating 5-81
- UPDATE_FAILED
 - DataStoreException class 2-42
 - ResolutionException class 5-89

- UPDATE_NOT_ALLOWED
 - ValidationException class 4-329
- UPDATED
 - RowStatus interface 4-262
- updated
 - EditListener interface 4-159
- updatedRow
 - ResolverListener interface 4-238
- updatedRowCount
 - StorageDataSet component 4-284
- updateError
 - EditListener interface 4-159
 - ResolverListener interface 4-238
- updateMode
 - QueryResolver component 5-83
- UpdateMode interface 4-319
- updateProcedure
 - ProcedureResolver component 5-53
- updateRow
 - DataSet class 4-112
 - ProcedureResolver component 5-55
 - QueryResolver component 5-85
 - SQLResolver class 5-102
- updating
 - EditListener interface 4-159
- updating data
 - in data sets 4-72
- updatingRow
 - ResolverListener interface 4-239
- URL_NOT_FOUND
 - DataSetException class 4-132
- URL_NOT_FOUND_IN_DESIGN
 - DataSetException class 4-132
- useCaseSensitiveId
 - Database component 5-19
- useCaseSensitiveQuotedId
 - Database component 5-19
- user input
 - parsing 4-39
 - restricting 4-41
- user responses 4-241
 - collecting 4-242
 - listener for 4-249
- userName
 - ConnectionDescriptor class 5-8
 - DataStoreConnection component 2-23
 - ServerConnection class 3-10
- useSchemaName
 - Database component 5-20
- useSetObjectForStreams
 - Database component 5-20
- useSetObjectForStrings
 - Database component 5-20
- useSpacePadding
 - Database component 5-20

- useStatementCaching
 - Database component 5-20
- useTableName
 - Database component 5-21
- useTransactions
 - Database component 5-21

V

- valid
 - Alignment class 6-7
- validate
 - Column component 4-53
 - ColumnChangeListener interface 4-56
 - DataRow class 4-85
 - DataSet class 4-112, 4-113
- validation
 - error handling 4-320
- ValidationException class 4-320
- ValidationException constructor
 - ValidationException class 4-330
- value
 - FastStringBuffer component 8-36, 8-47
- variant
 - Variant component 4-347
- Variant component 4-333
- Variant constructor
 - Variant component 4-340
- VariantException class 4-352
 - properties 4-352
- VariantException constructor
 - VariantException class 4-352
- VariantFormatStr class 6-74
- VariantFormatStr constructor
 - VariantFormatStr class 6-74, 6-75
- VariantFormatter class 6-80
- variantType
 - BigDecimalFormatter class 6-8
 - BinaryFormatter component 6-11
 - BooleanFormatter component 6-19
 - DateFormatter component 6-24
 - DoubleFormatter class 6-26
 - IntegerFormatter class 6-28
 - LongFormatter component 6-51
 - ObjectFormatter component 6-53
 - SimpleFormatter component 6-59
 - StringFormatter component 6-62
 - TimeFormatter component 6-70
 - TimestampFormatter component 6-72
 - VariantFormatStr class 6-76
 - VariantFormatter class 6-81
- VERBOSE
 - StreamVerifier class 2-52
- verify
 - StreamVerifier class 2-53

- version
 - DataStoreConnection component 2-23
- VERSION_3_0
 - DataStore component 2-12
- VERSION_4_0
 - DataStore component 2-13
- VERTICAL
 - Alignment class 6-6
- vetoableDispatch
 - EventMulticaster class 8-30
 - VetoableDispatch interface 8-52
- VetoableDispatch interface 8-52
- VetoException component 8-53
 - methods 8-54
- VetoException constructor
 - VetoException component 8-53
- vetoMessage
 - VetoException component 8-53
- views
 - providing alternate data set 4-144
- visible
 - Column component 4-50
- VSTRETCH
 - Alignment class 6-6

W

- warn
 - Diagnostic class 8-21, 8-22
- width
 - Column component 4-50

- write
 - AsciiOutputStream class 7-6
 - EncodedOutputStream class 7-11
 - FastBufferedOutputStream class 7-17
 - FileOutputStream class 2-46
 - FileStream class 2-48
 - SimpleCharOutputStream class 7-22
- WRITE_BLOCK_ERROR
 - DataStoreException class 2-42
- writeDelimited
 - SimpleCharOutputStream class 7-22
- writeln
 - SimpleCharOutputStream class 7-22, 7-23
- writeObject
 - DataStoreConnection component 2-32
- WRONG_DATABASE
 - DataSetException class 4-132
- wrongDatabase
 - DataSetException class 4-143

Y

- Y2K issues 1-3
- year 2000 issues 1-3