



digital
vision
group

TRIPnxp & TRIPjxp

Class Library Programmer's Guide

Version 8.4

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether "Software") supplied by DVG Operations GmbH (DVG) are exclusively owned by DVG.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of DVG. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by DVG to the extent specified in his contract with DVG.

The free-to-use non-commercial version doesn't require a prior written agreement with DVG but such customers, organizations and/or third parties agree by using the software and / or solution of DVG to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely unfringed and protected.



Table of Contents

INTRODUCTION	8
FILES AND LOCATIONS	8
<i>TRIPjxp for Java Programming</i>	8
<i>TRIPnxp for .Net Programming</i>	9
CODE SAMPLE FORMAT	9
PACKAGE / NAMESPACE CONVENTIONS	12
CLASS HIERARCHY	13
RECOMMENDATIONS REGARDING SEARCH AND RETRIEVAL	13
SERIALIZABLE OBJECTS	13
CONTROL OBJECTS	13
1. ERRORS AND EXCEPTIONS	15
2. SUCCESS MESSAGES	18
3. ESTABLISHING / TERMINATING A SESSION	20
LOCAL CONNECTIONS	23
TRIPNET CONNECTIONS	23
<i>Encrypted sessions</i>	24
WEB CONNECTIONS	25
GRID CONNECTIONS	25
AUTHENTICATION	25
<i>Logging into a physical connection</i>	25
<i>Authenticating with a TRIPgrid session</i>	26
<i>Logging out from a physical session</i>	26
TOKEN-BASED AUTHENTICATION	28
<i>Querying TRIPsystem Configuration</i>	28
<i>API Keys</i>	29
<i>Obtaining a Token Pair</i>	29
<i>Refreshing a Token</i>	30
<i>Revoking a Token Pair</i>	30
<i>Other Token Considerations</i>	31
ACCESSING THE SESSION	32
ACTIVITY LOGGING	32
4. PERFORMING CCL COMMANDS	34
INITIALIZING THE CCL COMMAND INTERPRETER	34
EXECUTING COMMANDS	37
RETRIEVING THE RESULT OF THE COMMAND	38
<i>Handling search history updates</i>	39
<i>Handling term lists</i>	40
<i>Handling hierarchical Display results</i>	41
<i>Handling output buffers</i>	44
NOTIFICATIONS	47
<i>The Notification Mechanism</i>	47
<i>Comforters</i>	48
TERM LISTS LOADED ON DEMAND	51
5. RETRIEVING DATA FROM DATABASES OR SEARCH SETS	54
PREPARING FOR RETRIEVAL	54
SEARCHED RETRIEVAL	55
REVERSE RETRIEVAL	56
SORTED RESULTS	57
DEFINING RESULT CONTENT	59
<i>Retrieving SString fields</i>	66
PROCESSING RESULTS USING TDBRECORD	67
PROCESSING RESULTS IN XML	72

Structure of the XML response	72
Transforming the result XML	74
Hit terms in the XML	77
Summary	77
6. TDBSEARCH	78
RATIONALE	78
<i>TdbCclCommand</i>	78
<i>TdbRecordSet</i>	78
Requirements	79
CREATING A TDBSEARCH INSTANCE	79
The <i>TdbSearch Class</i>	79
<i>TdbSearch Constructor</i>	79
EXECUTING CCL STATEMENTS	80
Overview	80
PERFORMING A SEARCH	81
API Summary	81
Search Example	84
FETCHING STRUCTURED DATA	84
Search Set Statistics	84
Record Cache	85
Record Retrieval	85
<i>TdbSearchSet</i> in for-each loops	86
Automatic Retrieval Templates	87
Assigning a Custom Retrieval Template	87
USING OUTPUT FORMATS	89
7. RETRIEVING DATA FROM CONTROL	90
CONTROL OBJECTS	90
CREATING AND USING CONTROL OBJECT LISTS	91
TRANSFORMING CONTROL OBJECT LISTS	95
DATABASE LIST PERFORMANCE CONSIDERATIONS	95
8. UPDATING DATABASES	97
CREATING OR RETRIEVING SINGLE RECORDS	97
MODIFYING OR ESTABLISHING THE CONTENT OF A TDBRECORD	101
Working with structured field types	104
Working with TExt fields	107
Working with STring fields	110
DELETING SINGLE RECORDS	113
AFFECTING MULTIPLE RECORDS WITH ONE REQUEST	114
Multiple insert	114
Multiple update	115
Multiple delete	115
CALLING ASE ROUTINES WHILE INSERTING OR UPDATING A RECORD	116
ASE List Properties on the <i>TdbRecord Class</i>	116
The <i>TdbCallAse Class</i>	117
Writing an ASE Routine	118
9. TUPLE LISTS	119
CREATING A TUPLE LIST	119
Specifying a tuple list using a field group	119
Specifying a tuple list explicitly	119
Ensuring presence of fields	119
NEW TUPLES	120
ACCESSING TUPLES	122
CLEARING TUPLES	123
REMOVING TUPLES	123

10. MANAGING DATABASES AND THESAURI	124
CREATING A NEW DATABASE OR THESAURUS	124
MODIFYING AN EXISTING DATABASE OR THESAURUS	128
<i>Deleting fields</i>	129
COPYING AN EXISTING DATABASE OR THESAURUS	130
DELETING EXISTING DATABASES AND THESAURI	131
11. MANAGING FORMATS	133
CREATING NEW FORMATS	133
MODIFYING EXISTING FORMATS	134
TESTING OUTPUT FORMATS	137
DELETING EXISTING FORMATS	139
12. MANAGING DATABASE CLUSTERS	140
CREATING A NEW CLUSTER	140
MODIFYING EXISTING CLUSTERS	141
DELETING EXISTING CLUSTERS	142
13. MANAGING CLASSIFICATION SCHEMES	143
CREATING A NEW SCHEME	143
MODIFYING AN EXISTING SCHEME	144
MANAGING CATEGORIES WITHIN A SCHEME	146
<i>Creating new categories</i>	147
<i>Retrieving existing categories</i>	147
<i>Training a category</i>	149
<i>Viewing training material for a category</i>	151
<i>Removing training for a category</i>	152
<i>Deleting an existing category</i>	153
TESTING CLASSIFICATION	153
USING AN EXISTING TRIP DATABASE TO CREATE AND TRAIN CATEGORIES	154
DELETING AN EXISTING CLASSIFICATION SCHEME	154
13. MANAGING ACCESS RIGHTS	155
WORKING WITH ACCESS RIGHTS	156
15. MANAGING USERS	159
CREATING NEW USERS	159
MODIFYING THE PROPERTIES OF AN EXISTING USER	161
DELETING EXISTING USERS	162
CHANGING OWNERSHIP	163
16. MANAGING USER GROUPS	165
CREATING NEW GROUPS	165
MODIFYING GROUP MEMBERSHIP	165
DELETING EXISTING GROUPS	166
CHANGING OWNERSHIP	166
17. MANAGING STORED PROCEDURES	169
CREATING NEW PROCEDURES	169
CREATE PROCEDURE BASED ON A SEARCH	170
MODIFYING EXISTING PROCEDURES	171
DELETING EXISTING PROCEDURES	174
18. CONNECTION POOLING	175
19. INTERACTION WITH TRIPCOF	181
API OVERVIEW	181
<i>Property TdbStringField:ExtractionTarget</i>	181
<i>Property TdbTextField:TextExtractionInfo</i>	182

Class <i>TdbTextExtractionInfo</i>	182
Method <i>TdbStringField.Convert</i>	188
EXTRACTING TEXT FROM A STREAM WITH SERVER-SIDE PROCESSING.....	190
EXTRACTING TEXT FROM A PREVIOUSLY STORED FILE DURING UPDATE.....	192
HTML CONVERSION.....	193
HTML CONVERSION WITH HTML HIGHLIGHTING.....	195
EXAMPLE OF CLIENT-SIDE HTML CONVERSION WITH HIGHLIGHTING.....	197
TEXT ANALYSIS.....	198
Class <i>TdbNlpInfo</i>	198
Text Analysis Example.....	203
20. USING JSON/XML DATABASES	204
OVERVIEW.....	204
API OVERVIEW.....	204
Enumeration <i>TdbXmlRecord.IOMode</i>	204
Enumeration <i>TdbXmlRecord.XmlRecordType</i>	205
Class <i>TdbXmlRecord</i>	205
INSERT XML DOCUMENTS.....	211
Inline import.....	211
Direct file import.....	212
Stream import.....	213
INSERT UNSTRUCTURED FILES IN JSON/XML DATABASES.....	214
Inline import.....	215
Direct file import.....	215
RETRIEVE DOCUMENTS FROM A JSON/XML DATABASE.....	216
Inline export.....	216
Direct file export.....	216
Stream export.....	217
UPDATE JSON/XML DATABASE RECORDS.....	218
USING XPATH QUERIES.....	219
Using XPath with the <i>TdbSearch Class</i>	219
Using XPath with the <i>TdbRecordSet Class</i>	219
Using XPath with the <i>TdbCclCommand Class</i>	220
RETRIEVING SEARCH RESULTS AS XML FRAGMENT SETS.....	221
21. FACETS	223
DEFINITION.....	223
FACETS IN TRIP.....	223
Facet Classes.....	223
USING TRIP FACETS.....	224
The <i>TdbFacetSet Class</i>	224
The <i>TdbFacetValue Class</i>	224
Facet Example.....	224
FACET BASELINES.....	225
Facet Baseline Registration Example.....	227
Facet Baseline Usage Example.....	228
Facet Scrolling Usage Example.....	229
22. GRAPHS	230
GRAPHS IN TRIP.....	230
Main Graph Classes.....	230
BASIC GRAPH OPERATIONS.....	231
Source, Target and Edge Sets.....	231
Forward Navigation.....	232
Backward Navigation.....	232
Resolving Vertices.....	233
Transitive Search.....	234
GRAPH PATH ANALYSIS.....	236
DEALING WITH LENGTHY PATH ANALYSIS OPERATIONS.....	238

<i>Session Timeout</i>	238
<i>Notificaitons</i>	239
THE TRIP GRAPH QUERY LANGUAGE	240
<i>Syntax</i>	240
<i>Graph Query APIs</i>	242
<i>Examples</i>	242
POPULATING A GRAPH	244
<i>Create a Graph Database</i>	244
<i>Adding an Edge</i>	245
<i>Adding a Vertex</i>	245
<i>Modifying a Graph Record</i>	246
23. CANCELLING COMMANDS	247



Introduction

This manual detail the means by which programmers can access TRIP using two class libraries of the TRIP SDK; *TRIPjxp* for Java development and *TRIPnxp* for development targeting the Microsoft .NET framework. Both of these class libraries are based on a new, high performance, XML-based network protocol called TRIPxpi. It exposes a number of atomic operations intended to collect what would traditionally be a large number of networked function calls into a single “transaction.”

TRIPxpi is therefore intended to be a high performance, low overhead network-oriented protocol suite that is well suited for application building in either middle-tier or even UI-tier (in the case of the TRIP AJAX toolkit) development.

In order to interact with TRIP using TRIPxpi, the programming model is exposed in two of the most prevalent programming environments, Java and Microsoft.Net. For the former, TRIPjxp provides a Java class library compatible with Java 8, 11 and 17, whilst for the latter, TRIPnxp provides a .Net 4.6.2-compliant assembly.

Each of these libraries expose the underlying functional model of TRIPxpi by encouraging the programmer to exercise a large number of local interactions with cached data coupled with a very small number of networked interactions. These networked interactions use the TRIPxpi protocols to request as much data and/or processing to be made available by the server as possible in a single request / response cycle.

For example, whilst the traditional TRIP programming model has centered around executing CCL searches and then using TRIP's report generator to create either intermediate or end user-consumable representations, scrolling through the kernel window buffers as necessary to collect all available data, TRIPxpi collects all of these various operations into a single request and response, encapsulating a search as well as the sorting and retrieval of results.

Although the amount of data involved in transmitting these request / response pairs can be significant, the overall bandwidth impact on individual servers and on the network as a whole is considerably diminished as compared to older libraries such as TRIPclient or TRIPcom, resulting in higher performance all around.

Application developers are very strongly encouraged to make use of the new protocols as described in section 5 (Retrieving data from databases or search sets) and section 6 (Retrieving data from CONTROL) in preference to the more traditional CCL command-centric interactions as described in section 4 (Performing CCL commands).

Files and locations

The class libraries are shipped using platform-appropriate packaging, as described below.

TRIPjxp for Java Programming

The main TRIPjxp library class files can found in tripjxp.jar. All example code can be found in tripjxp_examples.jar (executable) and tripjxp_examples_src.zip (Java source). In order to use TRIPjxp, simply ensure that tripjxp.jar is in your CLASSPATH.

For example, assuming that both tripjxp.jar and tripjxp_examples.jar are in the current directory, you can run any of the examples using the following type of command (also assumes that you have JAVA_HOME/bin in your execution path):


```
java -cp tripjxp_examples.jar;tripjxp.jar
    com.tietoenator.trip.jxp.examples.data.Transform
```

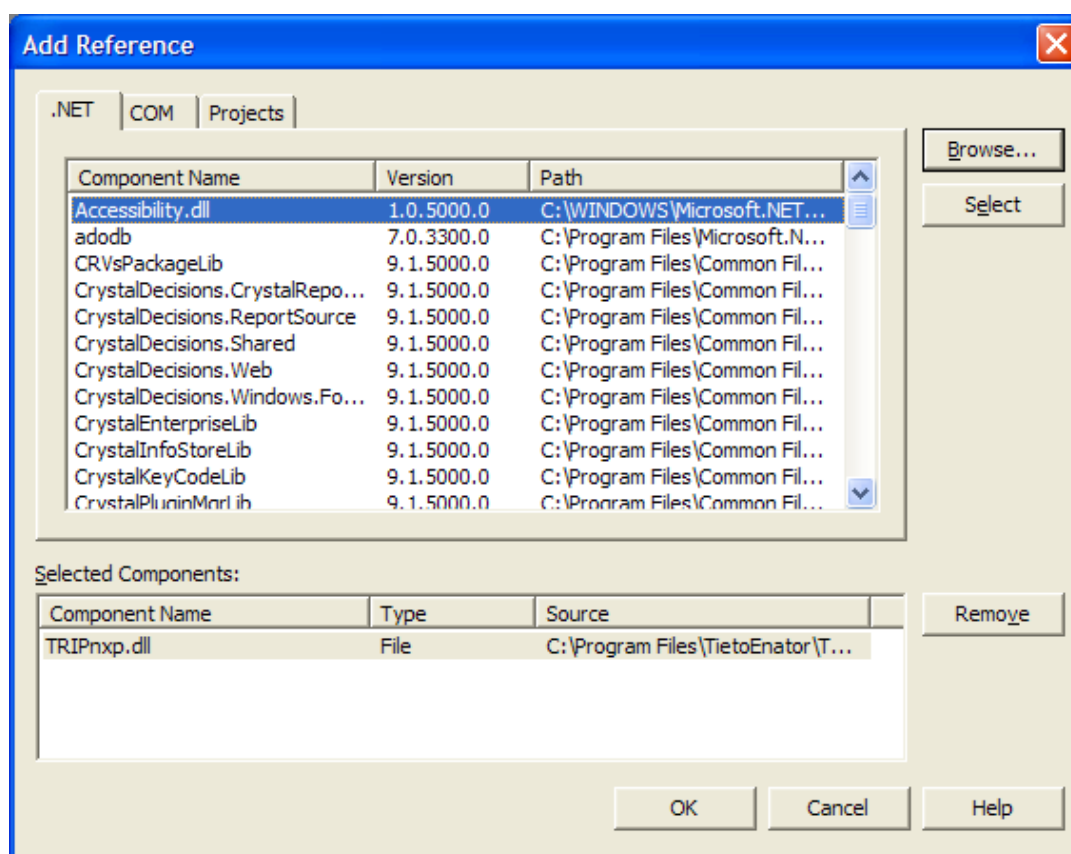
Using a development environment such as Eclipse, simply add the TRIPjxp JAR file to your workspace as normal.

Reference material (Javadoc) for TRIPjxp can be found in tripjxp_docs.zip. Reference material for the samples used in this guide can be found in tripjxp_examples_docs.zip.

TRIPnxp for .Net Programming

The TRIPnxp assembly, along with sample code and reference documentation, is shipped as a single Windows installation package. Install this package as usual, simply choosing a target directory for the entire installation.

To use TRIPnxp within a .Net project, simply add a reference to your project as usual:



Both the standard example application and the reference documentation are linked via a start menu folder entitled TietoEnator \ TRIPnxp. The source form of the examples can be found in the installation directory and can be opened as a project using the provided VB.Net project file.

Code sample format

There are three kinds of interaction that are common within both libraries: properties, collections, and methods. Strictly speaking collections are simply a different type of property, but for clarity we split them out in this guide. Also note that whilst the .Net library offers class indexers where appropriate, these are not explicitly documented in this guide. Information on indexers is included in the class library reference documentation.

When describing any of these interactions in detail, the format used for each will be similar and will reflect not only what the access mechanism is called, but what interaction types are legal.

Note that when describing .Net interactions, the data types shown will reflect the underlying system type (although undecorated with the System namespace for clarity). Programmers should obviously use the language-appropriate alias for the underlying type where applicable. The following common types are used in the library:

CLR Type	C# Type	VB.Net Type
System.String	string	String
System.Int32	int	Integer
System.Boolean	bool	Boolean
System.Single	float	Single
System.Byte	byte	Byte

For example, the following shows a read/write property of the TdbUser class:

Property: TdbUser:RealName

Type: String
Access: Read, Write

Java

```
String getRealName();
void putRealName(String name);
```

.Net

```
String RealName { get; set; }
```

Retrieve or establish the real name of the TRIP user.

A collection description would be as follows:

Collection: TdbDatabaseDesign:Fields

Type: List of TdbFieldDesign
Access: Read, Write

Java

```
List<TdbFieldDesign> fields()  
void putFields(Collection<TdbFieldDesign> fields)
```

.Net

```
ICollection Fields { get; set; }
```

Retrieve or establish the collection of field design templates for this database.

There are two types of collection used in the library, reflecting collections of items that are either ordered or not. Ordered collections are reflected as lists (List<E> or IList, respectively) whereas unordered collections are reflected simply as collections (Collection<E> or ICollection, respectively). In addition, many of the collections retrieved from the libraries are read only; programmers should take care to use language facilities to determine this access restriction before attempting to update collections retrieved from library properties.

Finally, a method description would be as follows (note the difference in capitalization between Java and .Net, reflecting the different coding standards for those platforms):

Method: TdbRecordSet:Get

Type: void
Throws: TdbException

Java

```
void get()
```

.Net

```
void Get()
```

Retrieve the underlying objects from the server. Applications must have set properties x, y and z prior to attempting to call this method, otherwise an INVALID_ARGS exception will be thrown.

When showing code fragments to explain the usage of a given method or property, again we give examples for the different platforms, for example as shown below.

Java

```
for( TdbFieldDesign field : myDatabase.fields() )
{
    String name = field.getName();
    ...
}
```

VB.Net

```
For Each field As TdbFieldDesign In myDatabase.Fields
    Dim name As String = field.Name;
    ...
Next
```

Note that code examples will not show different languages for the .Net platform as aside from language-specific syntax, the usage of the library is exactly the same between all CLR-compliant languages.

Package / Namespace conventions

The root package for the TRIPjxp library is:

```
com.tietoenator.trip.jxp
```

All classes are found either in this root package or in a child package, for example:

```
com.tietoenator.trip.jxp.database
```

Likewise for the .Net assembly, the root namespace is:

```
TietoEnator.Trip.Nxp
```

Child namespaces for particular functional areas are grouped appropriately, for example:

```
TietoEnator.Trip.Nxp.Users
```

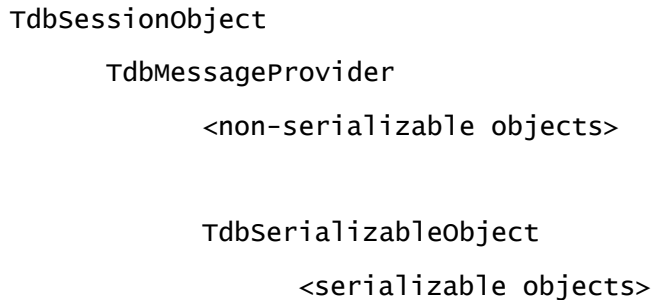
When describing a class location, this guide uses a common layout, as shown here:

Class: Tdbxyz

Derived from:	<base class>
Located in:	package / namespace location relative to root, or "root" if in the root package / namespace

Class hierarchy

There is a basic class hierarchy used within the library that helps ensure that all required information is available to users of class objects when required, as shown below.



Thus, when using an object of a class such as TdbDatabaseDesign, the class in question is derived from TdbSerializableObject, which in turn is derived from TdbMessageProvider and therefore from TdbSessionObject. This means that at all times, the capabilities offered by the lower level classes, such as retrieving the session to which the object is bound, or retrieving a list of available information messages, is freely available from the higher level object.

Very few classes within the library derive directly from an object other than shown in this hierarchy, and in such cases the classes in question are strict utility / container classes that afford no server interaction at all.

Recommendations regarding search and retrieval

Although both TRIPjxp and TRIPnpx provides more “traditional” means of search and retrieval by using classes for direct CCL command invocation and output formatting, the developer is encouraged to use the classes in the **data** package/namespace (e.g. TdbRecordSet, etc) instead whenever possible. These classes are specifically developed to address shortcomings in previous APIs that forced the developer to use suboptimal approaches, such as the use of output formats.

Serializable objects

One of the key levels in the class hierarchy is that of TdbSerializableObject. Any class derived from this base has a number of capabilities for serialization and deserialization, or in TRIP terms: export and import. Consult the reference documentation for the classes in question to see exactly how they expose this base functionality.

Control objects

Many of the classes within the library can be constructed using an instance of the class TdbControlObject as input. These “Control object references” are typically retrieved using the capabilities of the Control database retrieval classes documented in section 6 of this guide, and contain information from the Control database that uniquely reference the specific TRIP entity to which the calling program is binding the new object.

For example, a database entity within TRIP would be represented by a TdbControlObject with the following standard properties:

Name:	<database name>
Owner:	<name of the FM of the database>
CreateDate:	<date on which the design was created>
CreateTime:	<time at which the design was created>
ModifyDate:	<date on which the design was last modified>
ModifyTime:	<time at which the design was last modified>
Type:	Database (or thesaurus)
Comment:	<any description assigned to the database>

In addition, due to the object referenced being a database, the object will also have extended, or custom, properties such as:

RecordCount:	<number of records in the database>
ExtendedType:	<User, System, Demo, etc.>
Xml?:	<Yes / No>

Similarly, every entity type within the TRIP system can be represented using a `TdbControlObject`. This class provides a large number of properties and methods for interrogating the Control reference, allowing the programmer to determine exactly what it is, but equally the programmer can simply pass an object retrieved from Control (see section 6) into a constructor for an entity manipulation class, such as `TdbDatabaseDesign`, allowing for completely generic application construction.

Also note that all classes that can be constructed using a Control object reference implement an interface called `TdbControlObjectProvider`, and can themselves therefore be passed as construction arguments into any object that specifies a `TdbControlObject` as input.

1. Errors and exceptions

There are many different situations that can cause errors during a typical TRIP interaction. These errors can range from networking problems to syntax errors in commands to completely benign error conditions such as reaching the end of available data.

All types of errors and exceptions are reported using a single exception class:

Class: TdbException

Derived from: Exception
Located in: root

Every method of every class that throws an exception does so using an instance of this basic class. Instances of this class can be queried by the developer to retrieve either the textual form of the error message, or any assigned numeric error code.

As TdbException derives from the standard Exception class, it can be used in the normal fashion in exception catching clauses. In addition to the standard methods as required by Exception, TdbException defines the following properties that can be used to better understand, or report upon, the underlying problem.

Property: TdbException:Code

Type: Integer
Access: Read

Java

```
int getCode()
```

.Net

```
Int32 Code { get; }
```

Retrieve the underlying error code that triggered the exception being thrown. This is most likely to be a TRIP error message, but can also be a TRIPnet communication error code, etc.

Property: TdbException:DebugMessage

Type: String
Access: Read

Java

```
String getDebugMessage()
```

.Net

```
String DebugMessage { get; }
```

Retrieves a fully decorated message context for the underlying error, including a full stack trace of where the error occurred. This stack trace will include any nested exceptions (for example as thrown by operating system interface classes, etc.) that contributed in causing the exception to be thrown.

As all class methods can potentially throw TdbException instances, all invocations of methods must be wrapped in try/catch syntax, catching TdbException at the very least.

Outside of the range of normal TRIP error codes, the following TRIPxpi-specific error codes can be observed during TRIP interactions, these codes being provided to help the programmer in understanding context:

TdbException.UNEXPECTED_STATE

The client library is in an unexpected state, potentially due to a server glitch. The client program should probably restart at this point.

TdbException.COMMUNICATIONS_ERROR

An error occurred when attempting to communicate with the assigned TRIP server. The message accompanying this error code should help in determining the actual source of the problem.

TdbException.UNEXPECTED_RESPONSE

The server has responded to your request in a manner that is not consistent. This typically means a fatal error has occurred within the server's execution and any connection to that server should be reset in order to avoid further error conditions.

TdbException.PARSER_ERROR

The server has responded with invalid XML. This should never occur, but if it does it again points to a fatal error occurring within the server. The best recourse for the developer at this point is to reset the connection to the server.

TdbException.UNUSABLE_SESSION

The developer has attempted to invoke a method on a connection that cannot support that method's execution, for example requesting a grid-specific protocol on a TRIPnet connection.

TdbException.INVALID_ARGS

This error occurs when arguments passed to a method are invalid, for example null object references.

TdbException.END_OF_DATA

This is a benign exception code and reflects the program having successfully iterated to the end of a given collection, for example a particular field's content.

TdbException.VALIDATION_ERROR

This error occurs when internal state data of an object is invalid in the current context, e.g. null object references or non existing files to read. This typically means that some properties on the current object have not been set or have been assigned invalid values prior to the invocation of the method that threw this exception.

TdbException.NO_LICENSE

The current SDK product (TRIPnxp or TRIPjxp) is not licensed for use with the connected server.

TdbException.GENERAL_ERROR

An error was reported of an unhandled type, for example a heap corruption error when parsing XML, etc. Examine any nested exceptions for more detail – there will always be at least one nested exception provided.



2. Success messages

Many of the methods that cause TRIP server activity and that can reasonably be expected to have been generated by user interaction generate user-consumable messages indicating success, or information about what happened.

Classes that cause such success messages to be generated all extend a common base class that contains functionality to deal with, and to serve up, such messages:

Class: TdbMessageProvider

Derived from: TdbSessionObject
Located in: root

Thus, from any class that extends this base, the developer can retrieve any available success messages and/or message codes:

Collection: TdbMessageProvider:MessageList

Type: List of String
Access: Read

Java

```
List<String> getMessageList()
```

.Net

```
ICollection MessageList { get; }
```

Retrieve the collection of available information messages. Certain TRIPxpi interactions involve several, or many, TRIPapi functions being called on the server, each of which could feasibly generate a success message. In such cases, the success of the client method results in a list of success messages being made available to the calling application.

Collection: TdbMessageProvider:CodeList

Type: List of Integer
Access: Read

Java

```
List<Integer> getCodeList()
```

.Net

```
ICollection CodeList { get; }
```

Retrieve an iterator over the available list of success return codes.

Method: TdbMessageProvider:HasMessages

Type: Boolean
Throws: N/A

Java

```
boolean hasMessages()
```

.Net

```
Boolean HasMessages()
```

Check whether the object is in receipt of one or more success messages.

When the same object is being used for multiple operations, the list of messages should be reset to a blank state between iterations. The potential for user confusion is otherwise increased greatly, as the list of success messages would continue to grow with iterations. To reset any object of a class that extends TdbMessageProvider to a blank list of messages, use the following method:

Method: TdbMessageProvider:ResetMessages

Type: void
Throws: N/A

Java

```
void resetMessages()
```

.Net

```
void ResetMessages()
```

Clear any pre-existing list of success messages and/or codes.

3. Establishing / terminating a session

TRIPxpi access is available via several different physical connection mechanisms, such as local DLL linkage, TRIPnet, or HTTP. For networked interaction, although TRIPnet is the most efficient means of connecting directly to a single server, HTTP provides flexibility in terms of more generally-available network routing, and also provides support for communicating with TRIPgrid storage grids. Certain types of session can also be managed via a connection pool to help with multi-user performance, see chapter 16 for more details.

Objects of all classes, with the exception of the session classes described here, require an instance of a session class to be passed into their constructor, explicitly linking new objects to that session. Thus, the single most important first step in creating a TRIPxpi program is to create a session, and via that session to detail how the client will connect to the server for the purpose of exercising that session.

Class: TdbSession

Derived from: TdbSessionObject
Located in: session

Objects of this class cannot be created directly, however, as this base class does not provide a means of connection to a TRIP server. Instead, developers should create an object of one of the following types, depending on the transport desired:

Class: TdbLocalSession

Derived from: TdbSession
Located in: session

Class: TdbTripNetSession

Derived from: TdbSession
Located in: session

Class: TdbwebSession

Derived from: TdbSession
Located in: session

For TRIPgrid connections only, developers can create an object of the following type (see the appropriate sections of this document for the subset of protocols available on TRIPgrid connections – note that attempting to use non-grid aware protocols via a TRIPgrid connection will cause an UNUSABLE_SESSION exception to be thrown; likewise, attempting to use a grid protocol via a non-grid connection will cause the same exception):

Class: TdbGridSession

Derived from: TdbSession
Located in: session

The following code example shows how to create a session (we use TRIPnet in this example) and then login to that session.

Java

```
import com.tietoenator.trip.jxp.TdbException;
import com.tietoenator.trip.jxp.session.TdbSession;
import com.tietoenator.trip.jxp.session.TdbTripNetSession;

TdbSession init(String server, int port, String user, String pw)
{
    TdbSession session = null;
    try
    {
        session = new TdbTripNetSession(server, port);
        session.login(user, pw);
    }
    catch( TdbException e )
    {
        e.printStackTrace();
    }
    return session;
}
```

VB.Net

```
imports TietoEnator.Trip.Nxp;
imports TietoEnator.Trip.Nxp.Session;

Public Function init(ByVal server As String, _
                    ByVal port As Integer, _
                    ByVal user as String, _
                    ByVal pw As String) As TdbSession

    Dim session As TdbSession = nothing

    Try
        session = New TdbTripNetSession(server, port)
        session.Login(user, pw)
    Catch e As TdbException
        MsgBox e.Message
    End Try

    Return session

End Function
```

For an example of creating all kinds of session, please consult the following source examples

Java

```
com.tietoenator.trip.jxp.examples.util.SessionFactory
```

.Net

```
session\login.vb
```

The following sections describe each type of session connection and its available constructors. In general, connections that are either local (DLL linkage), or via TRIPnet or the TRIP web service are referred to as *physical*, versus connections to a TRIPgrid. Operations that are valid for any one type of physical connection are valid for all types of physical connection, but are typically not valid for a TRIPgrid connection (and vice versa, of course).

Local connections

This type of connection uses direct DLL linkage to access a TRIP installation on the local machine.

In order for this to work with TRIPnxp, the TRIP 'bin' folder must be in the user's PATH, or else the application and the assembly must be installed in the TRIP 'bin' folder.

For TRIPjxp, the use of local connections require that the 'tripjxp_local.jar' file is on the class path, and that the 'jxplocal' .dll/.so file is on the library path environment variable (PATH on Windows and LD_LIBRARY_PATH on most Linux/Unix systems). The library path can also be specified using the system property java.library.path.

There is only one constructor available for this type of connection (note that any attempt to construct an object of this class in Java will result in an UNUSABLE_SESSION exception being thrown):

Constructor: TdbLocalSession

.Net & Java

```
TdbLocalSession()
```

Create a connection to a TRIP installation running locally, using direct DLL linkage rather than any network protocol.

TRIPnet connections

There are four available constructors for TRIPnet connections:

Constructor: TdbTripNetSession

Java

```
TdbTripNetSession(String server, int port,  
                  String inifile, int timeout)
```

```
TdbTripNetSession(String server, int port)
```

```
TdbTripNetSession(String server, int port,  
                  String inifile, int timeout,  
                  boolean encrypt)
```

```
TdbTripNetSession(String server, int port,  
                  boolean encrypt)
```

```
.Net
```

```
TdbTripNetSession(String server, Int32 port,  
                    String inifile, Int32 timeout)
```

```
TdbTripNetSession(String server, Int32 port)
```

```
TdbTripNetSession(String server, Int32 port,  
                    String inifile, Int32 timeout,  
                    bool encrypt)
```

```
TdbTripNetSession(String server, Int32 port,  
                    bool encrypt)
```

Create a connection to the TRIPserver on the named server at the identified port. If specified, the named initialization file is requested to be executed by the server, and the connection timeout is set as specified (timeout is specified in milliseconds). If using the version of the constructor that does not specify timeout, a default timeout of 60 seconds is used.

Encrypted sessions

Support for encrypted sessions were added to version 7.0 of TRIPsystem and version 3.0 of TRIPnxp and TRIPjxp.

The TdbTripNetSession constructors that take a boolean "encrypt" parameter are used to establish encrypted sessions. Passing true to this parameter will result in an encrypted network communications channel being set up between the client process and the TRIP server.

The following steps are performed when establishing an encrypted session>

1. The client creates an asymmetric 1024 bit RSA key pair.
2. The public part of the RSA key is sent to the server along with a request to set up an encrypted session.
3. The server creates the session key. When used with TRIPsystem 8.2 or later, this is a 256-bit symmetric AES key. Older TRIPsystem versions creates a 192-bit symmetric Triple-DES encryption key.
4. The server encrypts the session key with the public part of the RSA key received from the client.
5. The encrypted key is sent to the client. Any further communications are hereafter assumed to be encrypted using the session key.
6. The client decrypts the session key using the private part of the RSA key it generated in step 1.

7. All communications for this session between the client and the server are hereafter encrypted.

Web connections

When connecting to the standalone TRIP web service, there is only one relevant constructor:

Constructor: TdbWebSession

Java

```
TdbwebSession(String server, int port)
```

.Net

```
TdbwebSession(String server, Int32 port)
```

Create a connection to the TRIP web service running on the named server at the identified port. The end point of this connection must be a servlet container with the TRIP web service deployed and active.

Grid connections

When connecting to the TRIPgrid web service, there is only one relevant constructor.

Constructor: TdbGridSession

Java

```
TdbGridSession(String server, int port)
```

.Net

```
TdbGridSession(String server, Int32 port)
```

Create a connection to the TRIPgrid web service running on the named server at the identified port. The end point of this connection must be a servlet container with the TRIPgrid web service deployed and active.

Authentication

Depending on the type of connection, there are two ways in which programs must identify themselves with the TRIP server. TRIPnet, local DLL linkage, and standalone web service connections are session-based and must therefore login to the target server. TRIPgrid connections, however, are not session-based and can therefore only authenticate a credentials set with the grid – note that if a grid session is not authenticated, any defined anonymous credentials as established by the grid DBA will be used by the grid web service when constructing slave TRIPnet sessions.

Logging into a physical connection

In order to login, the developer must provide a username and password, and can also choose to provide a language identifier and a restart flag.

Method: TdbSession:Login

Type: void
Throws: TdbException

Java

```
void login(String username, String password,  
           TdbLanguage language, boolean restart)
```

```
void login(String username, String password)
```

.Net

```
void Login(String username, String password,  
           TdbLanguage language, Boolean restart)
```

```
void Login(String username, String password)
```

Validate the combination of username and password provided (the password is always encrypted for transmission). If this succeeds, the resulting TRIP session is initialized to use the defined language as its CCL dialect, optionally restarting from a previously saved session file. If the version of the method is used that does not specify a language identifier, English is the default dialect.

Authenticating with a TRIPgrid session

In order to identify the calling process with the grid, the developer must use the following method call. Note that this is only legal on TRIPgrid sessions and results in the provided username and password being validated by the grid and then stored within the grid in order to identify the user in all interactions with slave TRIPnet connections.

Method: TdbSession:Authenticate

Type: void
Throws: TdbException

Java

```
void authenticate(String username, String password)
```

.Net

```
void Authenticate(String username, String password)
```

Dispatch the provided username and password to the TRIPgrid for authentication. The password is encrypted for transmission, as usual.

Note that there is no requirement to explicitly shut down, or terminate, a grid connection as all grid requests are atomic. Following 30 minutes of inactivity, however, any credentials authenticated with the grid will become invalid and further requests will operate within the context of the grid's anonymous user. If no such anonymous user has been established, all grid requests will fail until such time as the user is re-authenticated.

Logging out from a physical session

In order to explicitly terminate a physical connection, the following method instructs the server to shut down.

Method: TdbSession:Logout

Type: void
Throws: TdbException

Java:

```
void logout(boolean save)
```

```
void logout()
```

.Net

```
void Logout(Boolean save)
```

```
void Logout()
```

Instruct the server to shut down its connection with the TRIP server and to release any resources associated with that connection. The 'save' flag can be used to request the user's session file to be saved for later recovery during a subsequent login (this obviously requires a coherent site-wide policy with regards TDBS_SIF settings and session file storage).

If applications do not call this method, the server session will automatically terminate under the following circumstances:

Local	Upon process termination
TRIPnet	Upon client process termination
Web Service	After 30 minutes of inactivity

Token-based Authentication

Token based authentication is supported in TRIP from version 8.4. This commonly used type of authentication uses a pair of cryptographically secured tokens; one short-lived access token, and one longer-lived refresh token. Such a token pair can be obtained by any authenticated non-SYSTEM user. When the access token expires, the associated one-time use refresh token can be used to acquire a new token pair without having to authenticate with the user's username and password again.

The intended use case for the TRIP token-based authentication is to help TRIP based applications that use TRIP as their end user identity provider (i.e. the app users log on as TRIP users) implement an authentication scheme that can recall the user's identity over time, reducing the need for the user to explicitly log in with their username and password every time they wish to access the app. If returned from the web application server to the browser, the TRIP tokens should first be wrapped as JWT tokens. The exchange of such tokens is typically done via a cookie.

Note that despite access tokens elsewhere are mostly used for REST and stateless access, the TRIP token functionality is only an alternative means to authenticate (login). Authenticated TRIP sessions are always stateful, even if tokens are used.

This document will not go further into the details of token-based authentication for web applications; there are plenty of third-party resources for this purpose.

Querying TRIPsystem Configuration

Applications should use the `TdbSession` properties `[is]TokensEnabled` and `[get]ApiKeyMode` in order to determine if and to which extent token-based access is possible with the connected TRIPsystem server.

The `[is]TokensEnabled` property returns true if token-based access is enabled, and false if it is not.

Property: `TdbSession:TokensEnabled`

Type: `Boolean`
Access: `Read`

Java

```
boolean isTokensEnabled()
```

.Net

```
bool TokensEnabled { get; }
```

The `[get]ApiKeyMode` property returns an `TdbApiKeyMode` enum instance that denote if an API key is required for token use and, if so, for which operations.

Property: TdbSession:ApiKeyMode

Type: TdbApiKeyMode
Access: Read

Java

```
TdbApiKeyMode get ApiKeyMode();
```

.Net

```
TdbApiKeyMode ApiKeyMode { get; }
```

Retrieve the API key mode that the connected TRIPsystem server requires.

Enumeration: TdbApiKeyMode

Located in: session

API Keys

API keys, if enabled, provide an extra level of protection for token-based authentication. Not only must the application provide a valid access token, but also a valid API key. Such keys are generated by the TRIPsystem administrator and provided manually. There is no automatic way for applications to obtain such keys. API keys typically remain valid until the TRIPsystem administrator revokes them.

The possible values of the ApiKeyMode property are (with “Always” being the default unless otherwise configured in TRIPsystem):

- No** API keys are not required
- Always** An API key is required for the creation and refresh of tokens as well as for token-based login.
- Refresh** An API key is only required for the refresh of a token pair.
- Tokens** An API key is required for the creation and refresh of tokens, but not for token-based login.

Unless the ApiKeyMode is “No”, applications that use TRIP tokens must have a valid TRIPsystem API key. Make sure to store this key securely. Do NOT store the key in a source control repository or any other location where it is likely that it may be accessed by unauthorized parties!

Obtaining a Token Pair

If no refresh token is available or if it has expired, the application can request a new token pair on the behalf of the logged in user using the requestAccessToken method of the TdbSession class.

Method: TdbSession:RequestAccessToken

Type: TdbTokenPair
Throws: TdbException

Java

```
TdbTokenPair requestAccessToken(String apikey, String tag)
```

.Net

```
TdbTokenPair RequestAccessToken(String apikey, String tag)
```

This method returns a TdbTokenPair instance that contains the access and refresh token strings and information on their expiration times. The data in this instance should be securely stored and accessible by the associated end user only. If returned to the application web frontend, the tokens should first be JWT encoded.

The *apikey* argument must be a valid TRIPsystem API key if the ApiKeyMode of the session is "Always" or "Tokens".

The *tag* argument is an indicator to the TRIPsystem administrator regarding the intended use for the requested token. This can, for example, denote the application and/or app functionality that the TRIP sessions authenticated by the token will be used for. If you are unsure about what to specify here, ask your TRIPsystem administrator.

Refreshing a Token

An access token is typically rather short-lived, normally not longer than an hour. However, a refresh token can be valid for much longer (e.g. 30 days), but is one-time-use only. To use a refresh token to obtain a new token pair, the RefreshAccessToken method of the TdbSession class should be used.

Method: TdbSession:RefreshAccessToken

Type: TdbTokenPair
Throws: TdbException

Java

```
TdbTokenPair refreshAccessToken(String apikey,  
                                String refreshToken)
```

.Net

```
TdbTokenPair RefreshAccessToken(String apikey,  
                                String refreshToken)
```

This method returns a TdbTokenPair instance that contains the access and refresh token strings and information on their expiration times. The data in this instance should be securely stored and accessible by the associated end user only. If returned to the application web frontend, the tokens should first be JWT encoded.

The *apikey* argument must be a valid TRIPsystem API key if the ApiKeyMode of the session is "Always" or "Tokens".

The *refreshToken* argument must be a valid and as yet unused refresh token exactly as previously obtained via a call to RequestAccessToken or RefreshAccessToken.

Revoking a Token Pair

Revoking a token pair will permanently remove both the access token and its associated refresh token. No further authentication using this token will be possible. While

TRIPsystem administrators can unilaterally revoke tokens (e.g. as part of a scheduled key rotation, or due to a security leak), applications may in some cases also revoke tokens. Such revocation should be done whenever the application determines that the token pair will no longer be needed. Instead of letting the token pair fully expire by itself, it is in such cases more secure to explicitly revoke it.

Method: TdbSession:RevokeToken

Type: TdbTokenPair
Throws: TdbException

Java

```
TdbTokenPair revokeToken(String apikey, String token)
```

.Net

```
TdbTokenPair RevokeToken(String apikey, String token)
```

This method may be called without being logged in if a valid *apikey* is provided. If the session is in logged in state, the token must either be owned by the logged in user, or the logged in user must be the user manager (UM) of the user owning the token. Note that TRIPsystem may be configured to always require an API key for this operation, regardless of login status.

The *token* argument is expected to be the access or refresh token of the token to revoke.

Other Token Considerations

Each token pair requested for an end user via the RequestAccessToken method will have its cryptographically secured representation stored in TRIPsystem. A TRIP user may therefore have multiple tokens valid at the same time, where each one may be tagged for the same or a different purpose. However, if the application repeatedly keeps requesting a new access token for the same purpose without using the previous refresh token, the user will end up having a lot of unused token pairs whose refresh tokens remain valid. This is a serious security issue, should it happen!

Applications MUST therefore always use a refresh token if a valid one is available for the given purpose instead of requesting a new token pair from scratch. Ignoring this principle will make the TRIP installation less secure and give the TRIP administrator more work.

Accessing the session

In addition to passing a reference to the actual session object around, programs can make use of the fact that all objects in the library that interact with a session are derived from a common base class, `TdbSessionObject`, that exposes the session to which the object is linked via a property (as described in the Introduction section of this guide):

Property: `TdbSessionObject:Session`

Type: `TdbSession`
Access: `Read`

Java

```
TdbSession getSession()
```

.Net

```
TdbSession Session { get; }
```

Retrieve the `TdbSession` with which the object is associated, for use in constructing any other object that requires a valid session as input.

For example, the following code shows how objects can be constructed from each other.

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);
TdbFormatList ofs = new TdbOutputFormatList(db.getSession(), db);
```

VB.Net

```
Dim db as New TdbDatabaseDesign(session)
Dim ofs as New TdbOutputFormatList(db.Session, db)
```

Applications can also access the session's type, i.e. its connection type, using the `SessionType` property:

Property: `TdbSession:SessionType`

Type: `TdbInterfaceType`
Access: `Read`

Java

```
TdbInterfaceType getSessionType();
```

.Net

```
TdbInterfaceType SessionType { get; }
```

Retrieve the type of connection with which this session was created.

Enumeration: `TdbInterfaceType`

Located in: `session`

Activity logging

When creating or debugging applications using the libraries, it is often useful to see what the library is actually doing in response to the method calls being made. The class libraries

support logging of their TRIPxpi activity, i.e. all protocol requests and responses to and from the server, via the ActivityLog property of the TdbSession object.

Property: ActivityLog

Type: TdbActivityLogger
Access: Read, Write

Java

```
TdbActivityLogger getActivityLog()
void putActivityLog(TdbActivityLogger logger)
```

.Net

```
TdbActivityLogger ActivityLog { get; set; }
```

Establish or retrieve the instance of an activity logger that should be used to record all activity to/from the server.

This property uses classes that implement the TdbActivityLogger interface, all of which can be found in the “logging” package / namespace.

Class: TdbConsoleLogger

Derived from: Object
Located in: logging

Class: TdbFileLogger

Derived from: Object
Located in: logging

The first of these outputs all activity to the console whilst the latter outputs activity to the end of a named file. If neither of these methods is suitable for the application context, developers can simply implement the TdbActivityLogger interface on a class of their own to perform custom logging.

4. Performing CCL commands

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Also note that this section does not cover CCL syntax in any detail, although familiarity with the language is assumed in order to understand the examples. Full detail on CCL syntax can be found in the TRIP CCL Reference Guide.

For a complete example of interpreting and responding to CCL commands, consult the example applications:

Java

```
com.tietoenator.trip.jxp.examples.ccl.TRIPTty
```

.Net

```
Search\CclSearch.vb
```

Initializing the CCL command interpreter

Whenever a program has a need to execute a CCL command, either on behalf of the end user or for some internal purpose, the developer must create an instance of the following class:

Class: TdbCclCommand

```
Derived from:    TdbMessageProvider  
Located in:     ccl
```

Depending on the type of CCL command that is expected to be executed, the developer may choose to establish certain management structures that enable the retrieval of the results of those commands. Without these management structures in place, the display of results to an end user, for example, is impossible.

There are three types of management structure currently supported: kernel windows, term lists and term trees. The first of these is used to reflect the content of a kernel window buffer, i.e. the result of an output-generating command such as Show, SStatus, etc. The second is used to hold terms generated by a Display command, whilst the third is used to hold term hierarchies generated by a thesaurus-driven Display command.

The classes in question are:

Class: TdbKernelWindow

Derived from: TdbMessageProvider
Located in: ccl

Class: TdbTermList

Derived from: Vector<TdbTerm> (Java), ArrayList (.Net)
Located in: ccl

Class: TdbTermTree

Derived from: Object
Located in: ccl

After constructing a TdbCclCommand object, the developer should map any management structures that will be used before attempting to execute any CCL commands using that object. The following code extract shows how to establish a fully mapped object:



Java

```
import com.tietoenator.trip.jxp.TdbException;
import com.tietoenator.trip.jxp.ccl.TdbCclCommand;
import com.tietoenator.trip.jxp.ccl.TdbKernelWindow;
import com.tietoenator.trip.jxp.ccl.TdbKernelWindowType;
import com.tietoenator.trip.jxp.ccl.TdbTermList;
import com.tietoenator.trip.jxp.ccl.TdbTermTree;
import com.tietoenator.trip.jxp.session.TdbSession;

public TdbCclCommand init(TdbSession session) throws TdbException
{
    // Create the CCL command interpreter
    TdbCclCommand command = new TdbCclCommand(session);

    // Establish a term list/tree to hold Display command results
    command.setTermList(new TdbTermList());
    command.setTermTree(new TdbTermTree());

    // Map in a kernel window buffer for each type of output
    command.mapKernelWindow(new TdbKernelWindow(session,
        TdbKernelWindowType.Show));
    command.mapKernelWindow(new TdbKernelWindow(session,
        TdbKernelWindowType.Expand));
    command.mapKernelWindow(new TdbKernelWindow(session,
        TdbKernelWindowType.SysInfo));

    return command;
}
```

VB.Net

```
imports TietoEnator.Trip.Nxp;
imports TietoEnator.Trip.Nxp.Ccl;
imports TietoEnator.Trip.Nxp.Session;
```

```
Public Function init(ByVal session As TdbSession) As TdbCclCommand

    // Create the CCL command interpreter
    Dim command As New TdbCclCommand(session)

    // Establish a term list/tree to hold Display command results
    command.TermList = New TdbTermList
    command.TermTree = New TdbTermTree

    // Map in a kernel window buffer for each type of output
    command.MapKernelWindow(New TdbKernelWindow(session,
        TdbKernelWindow.Show));
    command.MapKernelWindow(New TdbKernelWindow(session,
        TdbKernelWindow.Expand));
    command.MapKernelWindow(New TdbKernelWindow(session,
        TdbKernelWindow.SysInfo));

    Return command

End Function
```

Notice that we create output window buffers for Show, Expand and SysInfo (this final window being used for commands like Help, SStatus, etc.) but not for the history or display window as we would in a normal TRIPapi application, as results for these windows are handled separately. This handling is described in the section on retrieving results, below.

Executing commands

In order to a CCL command, applications must use the following method. Note that as described above, in order to retrieve the results of any such command, the TdbCclCommand object being used must have been appropriately mapped to the various kernel window buffers and term list/tree structures.

Method: TdbCclCommand:ExecDirect

Type: void
Throws: TdbException

Java

```
void execDirect(String command)
```

.Net

```
void ExecDirect(String command)
```

Execute the command specified. Any result generated by the command must be retrieved from the object following successful completion. Any error condition, whether benign or fatal, will be thrown as a TdbException.

Retrieving the result of the command

Following successful execution of a command using the ExecDirect method, the results of the command may be retrieved using the various properties exposed by the TdbCclCommand class. In general, the first property used will be that to determine the type of operation that occurred on the server, so that appropriate processing may be performed:

Property: TdbCclCommand:CommandType

Type: Integer
Access: Read

Java

```
int getCommandType()
```

.Net

```
Int32 CommandType { get; }
```

This property yields a value from the TdbCclCommandType enumeration signifying the general type of command that was executed, and its general effect on the user's environment.

Enumeration: TdbCclCommandType

Located in: ccl

The values within this enumeration are as follows:

TdbCclCommandType:Misc

A command was executed that generated no resulting modification to the user's environment, except perhaps for one or more messages.

TdbCclCommandType:HistoryUpdate

One or more updates to the user's search history are available. See the section below on how to handle search results. Specifically, however, the results sent are to be appended to any UI representation of the user's search history.

TdbCclCommandType:HistoryReplace

The user's entire search history is included within the results. See the section below on how to handle search results. This command type is reflected in response to commands such as "list" or "renumber" or following the successful execution of a stored procedure.

TdbCclCommandType:TermList

The command generated a term list, i.e. a non-hierarchical Display command. The results will be contained within whatever term list structure was mapped to the CCL command object, as described in the section on initializing, above.

TdbCclCommandType:TermTree

The command generated a term tree, i.e. a hierarchical Display command using a thesaurus. The results generated will be contained within whatever term tree structure was mapped to the CCL command object, as described in the section on initializing, above.

TdbCclCommandType:Output

The command generated windowed output, e.g. a Show command. The output will be reflected in the appropriate TdbKernelWindow object as mapped into the CCL command object during initialization, as described in the initialization section, above. See the section below on retrieving window output for more detail.

TdbCclCommandType:Abort

The user has signaled a desire to stop doing whatever they're doing, using the "Stop" CCL command, either interactively or via a stored procedure.

Based on the value of the CommandType property, developers can create handler code that reflects the user's command appropriately through the UI, or simply updates internal structures if a UI is not part of the application context.

Handling search history updates

Several commands cause updates to, or indeed the wholesale replacement of, the user's search history. When the CommandType property signals that such a command has been interpreted, the program can retrieve whatever updates have been made using the following property:

Property: TdbCclCommand:HistoryUpdates

Type: List of TdbHistoryDetail
Access: Read

Java

```
Collection<TdbHistoryDetail> getHistoryUpdates()
```

.Net

```
ICollection HistoryUpdates { get; }
```

Retrieve the collection of search history updates generated by the most recent command.

The TdbHistoryDetail class contains methods to retrieve the ID of the search result generated, the number of records hit by the search, the number of terms hit by the search,

and the parsed form of the CCL command that generated the search result. From version 7.2-1, the name of the database associated with the search set is also provided. This information can be used to reflect a detailed search UI to end users, or simply for internal informational purposes.

Handling term lists

When the CommandType property signals that a term list has been updated, the information in the mapped term list (if any) is replaced with the newest information retrieved from the server. The currently mapped TdbTermList instance is retrieved from the command processor using the TermList property:

Property: TdbCclCommand:TermList

Type: TdbTermList
Access: Read, Write

Java

```
TdbTermList getTermList()  
void putTermList(TdbTermList list)
```

.Net

```
TdbTermList TermList { get; set; }
```

The appropriate TdbTermList object can then be queried to retrieve the terms for display or processing. The following example shows one way of handling Display term lists (note that in both cases, the “command” variable is assumed to be the active TdbCclCommand instance):

Java

```

case TermList:
{
    TdbTermList list = command.getTermList();
    System.out.println("" + list.size() + " terms");
    for( TdbTerm term : list )
        System.out.println("<" + term.getRecordCount() +
                           ">\t" + term.getTerm() + "\n");

    break;
}

```

VB.Net

```

Case TdbCclCommandType.TermList
    For Each term as TdbTerm in command.TermList
        myListBox.Items.Add(term.Term)
    Next

```

Handling hierarchical Display results

When the CommandType property signals that a term tree has been generated, the information in the mapped term tree (if any) is replaced with the newest information retrieved from the server. The currently mapped TdbTermTree instance is retrieved from the command processor using the TermTree property.

Property: TdbCclCommand:TermTree

Type: TdbTermTree
Access: Read, Write

Java

```

TdbTermList getTermTree()
void putTermTree(TdbTermTree tree)

```

.Net

```

TdbTermTree TermTree { get; set; }

```

As with the term list, the application can then use properties of the retrieved TdbTermTree to represent the tree in some consumable fashion.

Collection: TdbTermTree:Roots

Type: List of TdbTreeEntry
Access: Read

Java

```
List<TdbTreeEntry> roots()
```

.Net

```
ICollection Roots { get; }
```

Retrieve the set of tree nodes that are most generic, i.e. which have no parent nodes, or more generic terms. Each node retrieved is the root of a tree, the first generation of which can be retrieved using the Children property of each root node.

Property: TdbTermTree:RootCount

Type: Integer
Access: Read

Java

```
int getRootCount()
```

.Net

```
Int32 RootCount { get; }
```

Retrieve the number of nodes in the tree that are root nodes, i.e. which have no parent nodes, or more generic terms. Applications could also use the size() or Count of the Roots property to save traversing the tree twice for the same information.

The following examples show ways of interacting with the TdbTermTree.

Java

```

case TermTree:
{
    TdbTermTree tree = command.getTermTree();
    System.out.println("'" + tree.getRootCount() + " trees.");
    for( TdbTreeEntry node : tree.roots() )
        showTree(node, 0);
}

...

// Simply print out nodes from the tree, using spaces
// to indent child levels from parent levels
static final String spaces = "                ";
void showTree(TdbTreeEntry node, int indent)
{
    System.out.print(spaces.substring(0, indent));
    System.out.println(node.getName());
    for( TdbTreeEntry child : node.children() )
        dumpTree(child, indent + 2);
}

```

VB.Net

```

Case TdbCclCommandType.TermTree
    Dim tree as TdbTermTree = command.TermTree
    For Each node As TdbTreeEntry In tree.Roots
        addNodeToTree(node, Nothing)
    Next
...

```

```
' Recursively construct a Treeview control's node list from
' the term tree
Public Sub addNodeToTree(ByVal node as TdbTreeEntry, _
                        ByVal parent as TreeNode)

    Dim entry as new TreeNode(node.Name)

    If parent Is Nothing Then
        myTree.Nodes.Add(entry)
    Else
        parent.Nodes.Add(entry)
    End If

    For Each child As TdbTreeEntry In node.Children
        addNodeToTree(child, entry)
    Next

End Sub
```

Handling output buffers

Various CCL commands result in output being made available by the output formatter. This can be output from the currently open database, or a generated summary of statistics, or information from the CONTROL data dictionary, etc. All such output, however, is handled in exactly the same way by TRIPxpi.

Upon being signaled that output has been generated, applications should query the TdbCclCommand object to determine which window buffer was affected using the AffectedWindow property.

Property: TdbCclCommand:AffectedWindow

Type: TdbKernelWindow
Access: Read

Java

```
TdbKernelWindow getAffectedWindow()
```

.Net

```
TdbKernelWindow AffectedWindow { get; }
```

Retrieve the most recently affected TdbKernelWindow object. From that object applications can retrieve the content of the window buffer and any hit points that might be relevant.

The following sample shows one simple way of handling window output:

Java

```
case output:
    System.out.println(command.getAffectedWindow().toString());
    break;
```

VB.Net

```
Case TdbCclCommandType.Output
    myEditControl.Text = command.AffectedWindow.ToString();
```

In this example, hit terms are ignored. In a more robust application, of course, hit terms are important and should be reflected in any output generated for end users.

For the .Net platform only, the ToRtfString method creates valid Rich Text Format from the kernel window buffer, including marking up hits in a different font than normal text.

Method: TdbKernelWindow:ToRtfString

Type: String
Throws: TdbException

Java

Not available

.Net

```
String ToRtfString(TdbKernelWindowFont normalText,
                  TdbKernelWindowFont hitpointText)
```

This method returns an RTF-compliant text string suitable for viewing with the .Net RTF viewer control (or any other RTF-compliant application, e.g. WordPad / Word / etc.). The two fonts provided are used to output normal and hit point text, respectively.

Class: TdbKernelWindowFont

Derived from: Object
Located in: ccl

The following example shows how to use this method.

VB.Net

```
' Create two fonts for normal text and hit terms; the default
' font attributes, which can be overridden using properties of
' the TdbKernelWindowFont class, are Lucida Console, 10pt, normal
' (or whatever the platform equivalent is, depending on the
' windows version installed)
Dim normal As New TdbKernelWindowFont
Dim hit As New TdbKernelWindowFont

' Define hits as being shown in red
hit.Red = 255

' Send the RTF to a standard RTF viewer control
myRtfControl.Rtf = command.AffectedWindow.ToRtfString(normal, hit)
```

For both Java and .Net, the ToFormattedString method can be used to create a highlighted rendition of the window's content, including inserting arbitrary markup before and after hit terms:

Method: TdbKernelWindow.ToFormattedString

Type: String
Throws: TdbException

Java

```
String toFormattedString(String before, String after,
                          String linebreak)
```

.Net

```
String ToFormattedString(String before, String after,
                          String linebreak)
```

Create a string representation of the window's content, inserting arbitrary markup before and after hit terms, for example "" and "" in the simplest of HTML cases. The linebreak argument can be used to arbitrarily delimit lines within the buffer, e.g. "\n" for text, etc.

The following example assumes that the output format in question is actually outputting HTML and that all that the program wishes to do is to emit SPAN tags around hit points. In this context, emitting line-break sequences would be unwarranted, of course:

Java

```
TdbKernelWindow w = command.getAffectedWindow();
String before = "<span class=\"hitpoint\">";
String after = "</span>";
String content = w.toFormattedString(before, after, "");
```

VB.Net

```
Dim w As TdbKernelWindow = command.AffectedWindow
Dim before As String = "<span class=\"hitpoint\">"
Dim after As String = "</span>"
Dim content As String = w.ToFormattedString(before, after, "")
```

Most generally, the text lines and hit positions within the buffer are available using the Lines and Hits collections.

Collection: TdbKernelWindow:Lines

Type: List of String
Access: Read

Java

```
List<String> getLines()
```

.Net

```
IList Lines { get; }
```

Retrieve the collection of lines that comprise the buffer's contents.

Collection: TdbKernelWindow:Hits

Type: List of TdbHitPoint
Access: Read

Java

```
List<TdbHitPoint> getHits()
```

.Net

```
IList Hits { get; }
```

Retrieve the collection of hit points for the buffer's current contents.

Note that as kernel window buffers are tied to the capabilities of TRIP's output formatter, the recommended method of generating rich output such as is required for modern web applications is to use the data retrieval capabilities described in the next section.

Notifications

The Notification Mechanism

Notifications is a mechanism introduced in TRIPsystem 6.2-8 for which APIs were added to TRIPjxp and TRIPnpx in version 2.1-1. This mechanism is a means by which the server can inform the client of various events as it is processing a client request.

In order to receive notification messages, the application must subclass the `TdbNotificationSink` class and assign an instance of it to the `NotificationSink` property of the active `TdbSession`.

Class: `TdbNotificationSink`

Derived from: `TdbMessageProvider`
Located in: `session`

Property: `TdbSession.NotificationSink`

Type: `TdbNotificationSink`
Access: Read/Write

Java

```
void setNotificationSink(TdbNotificationSink sink)
TdbNotificationSink getNotificationSink()
```

.Net

```
TdbNotificationSink NotificationSink { get; set; }
```

Comforters

The notification types currently supported are search and sort comforters. These notifications can be sent by the server when it is processing queries and sort orders that take a long time to complete. To receive comforter notifications, the application must override the methods `onSearchComforter` and/or `onSortComforter` in its `TdbNotificationSink` subclass.

Method: `TdbNotificationSink.OnSearchComforter`

Type: `boolean`
Throws: `TdbException`

Java

```
boolean onSearchComforter(String message)
```

.Net

```
bool onSearchComforter (String message)
```

Return true from `onSearchComforter` to tell the server to continue processing the query, and false to abort it.

Method: TdbNotificationSink:OnSortComforter

Type: boolean
Throws: TdbException

Java

```
boolean onSortComforter(String message)
```

.Net

```
bool OnSortComforter (String message)
```

Return true from onSortComforter to tell the server to continue sorting, and false to abort it.

Java: Notification handler

```
public class MyComforter extends TdbNotificationSink
{
    public MyComforter(TdbSession session)
    {
        super(session);
    }
    public boolean onSearchComforter(String message)
    {
        System.out.println(message);
        return true;
    }
}
```

Comforter signaling is not enabled by default in the server. So in addition to subclassing `TdbNotificationSink` and overriding one or more of its methods, the application must also call the `TdbSession.enableNotification` method.

The application can also disable comforter signaling by calling `TdbSession.enableNotification(TdbNotificationType,boolean)` with 'false' as the the second argument .

Method: TdbNotificationSink:EnableNotification

Type: void
Throws: TdbException

Java

```
void enableNotification(TdbNotificationType,  
                        int interval)  
  
void enableNotification(TdbNotificationType,  
                        boolean enable)
```

.Net

```
void EnableNotification(TdbNotificationType,  
                        int interval)  
  
void EnableNotification(TdbNotificationType,  
                        bool enable)
```

The first version of the EnableNotification method enables notifications of a certain type for the specified number of seconds. The second version of the method enables or disables notifications of a specified type. If the second version of the method is used to enable notifications, a default interval of 5 seconds will be used.

The example below enables comforter signaling with an interval of four seconds. Any search or sort order that takes longer than four seconds will after this cause the notification sink to be invoked.

Java

```
session.setNotificationSink(new MyComforter(session));  
session.enableNotification(TdbNotificationType.COMFORTER,4);
```

.NET

```
session.NotificationSink = new MyComforter(session);  
session.EnableNotification(TdbNotificationType.COMFORTER,4);
```

Comforter notifications can be sent regardless of the origin of the search or sort order. Search and sort orders sent via the classes TdbCclCommand, TdbRecordSet and TdbSearch all can cause notifications to be sent. This also applies to search and sort orders that are part of TRIP procedures or executed in the context of an Application Software Exit (ASE) routine.

Term lists loaded on demand

When using TRIPjxp and TRIPnpx 3.0 or later with TRIPsystem 7.0 or later, term lists that are the result of DISPLAY orders can be retrieved on demand and "streamed" from the client in blocks of 100 terms each. This behavior reduces processing and I/O wait time when generating very large term lists, but may add extra some extra network I/O with small term lists.

The on-demand term list feature can be enabled using the UseOnDemandTermLists property which is available on both the TdbSearch and TdbCclCommand classes.

Property: TdbSearch:UseOnDemandTermLists
TdbCclCommand:UseOnDemandTermLists

Type: Boolean
Access: Read/Write

Java

```
boolean getUseOnDemandTermLists()
void setUseOnDemandTermLists(boolean enable)
```

.Net

```
bool UseOnDemandTermLists { get; set; }
```

Check if on-demand term lists are enabled, and enable/disable on-demand term lists. If on-demand term lists cannot be enabled, the setter method for this property will throw a TdbException.

You can use several term lists concurrently provided they are in on-demand mode. You can check if this is so using the property IsOnDemand on the TdbTermList class:

Property: TdbTermList:IsOnDemand

Type: Boolean
Access: Read

Java

```
boolean isOnDemand()
```

.Net

```
bool IsOnDemand { get; }
```

If the term list is in on-demand mode, the terms in the list will be retrieved as needed. No terms at all will be retrieved until the application requests a term from the list.

On-demand term lists have a timeout such that the associated server-side resources get removed automatically after 120 seconds. This timeout is in place in order to protect against overuse of server resources. This timeout period can be extended or even disabled using the `TermListTimeout` property on the `TdbCclCommand` and `TdbSearch` classes.

Property: `TdbSearch:TermListTimeout`
`TdbCclCommand:TermListTimeout`

Type: Boolean
Access: Read/Write

Java

```
int getTermListTimeout ()  
void setTermListTimeout (int seconds)
```

.Net

```
int TermListTimeout { get; set; }
```

This property determines the life span of on-demand term lists. Set to zero in order to disable the timeout for the next DISPLAY order that you intend to access using the on-demand term list feature.

In order to keep an on-demand term list active while you are issuing a new DISPLAY command, you must assign a new `TdbTermList` instance to the `TdbSearch` or `TdbCclCommand` instance you used to generate the first term list. See example below (error handling omitted for clarity):

Java

```
TdbCclCommand cmd = new TdbCclCommand(session);  
cmd.execDirect("BASE ALICE");  
cmd.setUseOnDemandTermLists(true);  
  
cmd.execDirect("DISPLAY PERSON=#");  
TdbTermList firstList = cmd.getTermList();  
cmd.setTermList(new TdbTermList());  
  
cmd.execDirect("DISPLAY SPEAKER=#");  
TdbTermList secondList = cmd.getTermList();  
cmd.setTermList(new TdbTermList());  
  
// work with the lists, then close() them when you're done  
  
firstList.close();  
secondList.close();
```

VB.Net

```
Dim cmd As TdbCclCommand = new TdbCclCommand(session)
cmd.execDirect("BASE ALICE")
cmd.UseOnDemandTermLists = true;

cmd.execDirect("DISPLAY PERSON=#")
Dim firstList as TdbTermList = cmd.TermList
cmd.TermList = new TdbTermList()

cmd.execDirect("DISPLAY SPEAKER=#")
Dim secondList as TdbTermList = cmd.TermList
cmd.TermList = new TdbTermList()

' work with the lists, then close() them when you're done

firstList.close()
secondList.close()
```

5. Retrieving data from databases or search sets

Note that this chapter relates physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

As mentioned before, developers are strongly encouraged to consider developing new applications using the data retrieval classes and methods described in this section, rather than using the CCL command interpreter described in section 4, where appropriate.

Many applications can be boiled down to, at their simplest, a cycle of search, view result list, view one or more documents in their entirety, and repeat. This cycle is encapsulated within the data retrieval classes in its most efficient form, performing searches, sorting results and retrieving data from hit records, all within the context of a single server request.

The basis for all such operations is the TdbRecordSet class:

Class: TdbRecordSet

Derived from: TdbMessageProvider
Located in: data

This class defines the notion of a set of records extracted from a database or search set, each record comprising a set of components, each component comprising a set of fields, each field comprising a set of values.

The record set defines a series of operations to determine its eventual function, although only one method actually involves any network interaction with the server.

To take a simple example, assume that an application wishes to follow the basic order of operation discussed here:

- Accept search criteria from the end user
- Perform the search, report on the number of records hit
- Sort the results, according to relevance or other data-driven keys
- Retrieve a result set consisting of titles and focused hits from the textual content of the record
- Present this result set to the end user via some UI mechanism such as a web browser

Using the TdbRecordSet class, this entire cycle can be performed in a single network operation, and the results retrieved can be processed in a number of different ways, as described below.

Preparing for retrieval

In order to retrieve data, we have to prepare the record set in terms of where it should be retrieving data from, what criteria the data must match in order to be retrieved, in what order records should be retrieved, etc.

For sake of example, assume that we wish to retrieve the first 5 records from the database ALICE:

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setDatabase("alice");
rs.setFrom(1);
rs.setTo(5);
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)
rs.Database = "alice";
rs.From = 1;
rs.To = 5;
rs.Get();
```

The basic networked operation here is the Get method, which retrieves whatever range of records have been requested, from a database, a search set, or the results of a query statement. All the properties on the record set exist to allow the developer to state the nature of the retrieval request, but nothing is sent to the server until the Get method is invoked.

Method: TdbRecordSet:Get

Type: void
Throws: TdbException

Java

```
void get()
```

.Net

```
void Get()
```

Send a request for information to the server. The records retrieved are deserialized into a collection of TdbRecord instances.

Searched retrieval

The most common type of data retrieval request, of course, is that following a query. In support of this requirement, TdbRecordSet allows for the statement of queries as the bounding domain of a data retrieval operation, using the normal CCL dialect.

Extending the example from above, assume that instead of retrieving the first 5 records from the database ALICE, we want to retrieve the first 5 results for the search "Find mad hatter" within the database ALICE.

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setDatabase("alice");
rs.setQuery("find mad hatter");
rs.setFrom(1);
rs.setTo(5);
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)
rs.Database = "alice"
rs.Query = "find mad hatter"
rs.From = 1
rs.To = 5
rs.Get()
```

Reverse Retrieval

From version 1.2 of TRIPnpx and TRIPjxp it is possible to specify retrieval of the record set in reverse. This is equivalent of the CCL command "SHOW REVERSE", but will work on sorted retrieval as well. The range of records to retrieve (the From and To properties) here specify ordinal numbers of the reversed set, i.e. number 1 represents the last record in the unreversed set.

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setDatabase("alice");
rs.setFrom(1);
rs.setTo(5);
rs.setRetrieveReversed(true);
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)
rs.Database = "alice"
rs.From = 1
rs.To = 5
rs.RetrieveReversed = true;
rs.Get()
```


Following such a query operation, applications have access to extra information about the query results:

Property: TdbRecordSet:QueryRecords

Type: Integer
Access: Read

Java

```
int getQueryRecords()
```

.Net

```
Int32 QueryRecords { get; }
```

Retrieve the total number of records hit by the search; the records retrieved by the operation will be a subset of this total.

Property: TdbRecordSet:QueryHits

Type: Integer
Access: Read

Java

```
int getQueryHits()
```

.Net

```
Int32 QueryHits { get; }
```

Retrieve the total number of term occurrences hit by the search.

Sorted results

By default, records are retrieved from databases and search sets in database order, i.e. sorted simply by record number. This behavior is easily modified to specify arbitrary sort sequences, as shown by this modification of the example used above.

Java

```
TdbRecordSet rs = new TdbRecordSet(session);  
rs.setDatabase("alice");  
rs.setQuery("find mad hatter");  
rs.setFrom(1);  
rs.setTo(5);  
rs.setSortKeys("speaker, chaptnr");  
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)  
rs.Database = "alice"  
rs.Query = "find mad hatter"  
rs.From = 1  
rs.To = 5  
rs.SortKeys = "speaker, chaptnr"  
rs.Get()
```

The string specified as the SortKeys property can contain any collection of valid field names from the database(s) set as the domain of the retrieval request.

Applications can also request results to be sorted by relevance rank. Note that relevance rank is always used as the primary sort key if specified, and is always sorted in descending order. Applying relevance ranked sorting is shown in this modification of the example used above:

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setDatabase("alice");
rs.setQuery("find mad hatter");
rs.setFrom(1);
rs.setTo(5);
rs.setSortKeys("speaker, chaptnr");
rs.setSortRanked(true);
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)
rs.Database = "alice"
rs.Query = "find mad hatter"
rs.From = 1
rs.To = 5
rs.SortKeys = "speaker, chaptnr"
rs.SortRanked = True
rs.Get()
```

Defining result content

Thus far, we have defined queries and parameters but we have yet to request any actual result record contents. Without record content being requested, the server will execute the query and return the summary results (i.e. number of records, number of hits, success/failure messages, etc.), but will not return any actual data.

To request data to be retrieved, the calling application must define a template to which all records retrieved should conform. This template is defined in terms of components and fields, and can be as broad or as specific as required.

To create a template, the application must create a TdbRecord instance, and then add fields and component definitions to that record's template. For example:

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
TdbRecord tmp1 = new TdbRecord(session);

// Create a simple template: the fields "chapter" and "speaker"
// are to be retrieved in their entirety; the field "txt" is to
// be retrieved in focused form, showing context around hit terms
tmp1.addToTemplate("chapter");
tmp1.addToTemplate("speaker");
tmp1.addToTemplate(new TdbFieldTemplate("txt", true, false, 100));

// Link the template to the record set
rs.setRetrievalTemplate(tmp1);

// Set any other properties of the record set
...

// Retrieve the records from the server
rs.get();
```

VB.Net

```

Dim rs As New TdbRecordSet(session)
Dim tmp1 As New TdbRecord(session)

' Create a simple template: the fields "chapter" and "speaker" are
' to be retrieved in their entirety; the field "txt" is to be
' retrieved in focused form, showing context around hit terms
tmp1.AddToTemplate("chapter");
tmp1.AddToTemplate("speaker");
tmp1.AddToTemplate(New TdbFieldTemplate("txt", true, false, 100));

// Link the template to the record set
rs.RetrievalTemplate = tmp1;

// Set any other properties of the record set
...

// Retrieve the records from the server
rs.Get()

```

Depending on how the template addition is constructed, various different retrieval options can be specified.

Method: TdbRecord:AddToTemplate

Type: void
Throws: N/A

Java

```

void addToTemplate(String name)

void addToTemplate(TdbFieldTemplate fieldTemplate)

```

.Net

```

void AddToTemplate(String name)

void AddToTemplate(TdbFieldTemplate fieldTemplate)

```

Add a field request to the retrieval template. Using the simple version adds a request for the field in its entirety. For more options, create an instance of the TdbFieldTemplate class to pass in to the method. If the field specified is a member of the part record structure, then all parts are retrieved unless otherwise restricted.

Class: TdbFieldTemplate

Derived from: Object
Located in: data

Constructor: TdbFieldTemplate*Java*

```
TdbFieldTemplate(String name)
```

```
TdbFieldTemplate(String name, boolean focused,  
                  boolean summarized, int size_hint)
```

.Net

```
TdbFieldTemplate(String name)
```

```
TdbFieldTemplate(String name, Boolean focused,  
                  Boolean summarized, Int32 size_hint)
```

Construct a field request that can be added to a record set's retrieval template. Using the simple version, the field is requested in its entirety. Further options on the fully specified version of the constructor allow the request to specify whether the field's content should be focused on search terms, should be summarized, and if so how much text should be returned at a minimum.

In order to mark hits within retrieved fields, applications can either set field-specific markup using properties of each TdbFieldTemplate instance added to the retrieval template, or can establish a common set of markup strings for all fields in the template, once those fields have been added.

Property: TdbFieldTemplate:BeforeHits

Type: String
Access: Write

Java

```
void setBeforeHits(String markup)
```

.Net

```
String BeforeHits { set; }
```

Set the markup that should be emitted before hit points that occur in this specific field.

Property: TdbFieldTemplate:AfterHits

Type: String
Access: write

Java

```
void setAfterHits(String markup)
```

.Net

```
String AfterHits { set; }
```

Set the markup that should be emitted after hit points that occur in this specific field.

Method: TdbRecord:SetTemplateMarkup

Type: void
Throws: N/A

Java

```
void setTemplateMarkup(String before, String after)
```

.Net

```
void SetTemplateMarkup(String before, String after)
```

Establish markup strings that will be emitted before and after hit terms in all fields that have been added to the retrieval template to date. Fields added to the template after calling this method will not receive the markup specified here.

The following example shows how these various properties and methods might be used to show different types of hit term markup in different fields within a request.

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
TdbRecord template = new TdbRecord(session);

// Add three fields, all of which will receive the same markup
template.addToTemplate("chapter");
template.addToTemplate("speaker");
template.addToTemplate("person");

// Set the markup that fields added so far will receive
template.setTemplateMarkup(">>", "<<");

// Now add a fourth field with field-specific markup
TdbFieldTemplate fld = new TdbFieldTemplate("txt",true,false,100);
fld.setBeforeHits("***");
fld.setAfterHits("***");
template.addToTemplate(fld);

// Set other record set parameters
...

// Now retrieve the record set
rs.setRetrievalTemplate(template);
rs.get();
```


VB.Net

```
Dim rs As New TdbRecordSet(session)
Dim template As New TdbRecord(session)

' Add three fields, all of which will receive the same markup
template.AddToTemplate("chapter")
template.AddToTemplate("speaker")
template.AddToTemplate("person")

' Set the markup that the fields added so far will receive
template.SetTemplateMarkup(">>", "<<")

' Now add a fourth field with field-specific markup
Dim fld as New TdbFieldTemplate("txt", true, false, 100)
fld.BeforeHits = "***"
fld.AfterHits = "***"
template.AddToTemplate(fld)

' Set of other record set parameters
...

' Now retrieve the record set
rs.RetrievalTemplate = template
rs.Get()
```

To restrict the components that are retrieved, calling applications can set additional properties of the template record.

Property: TdbRecord:RetrieveHead

Type: Boolean
Access: Write

Java

```
void setRetrieveHead(boolean mode)
```

.Net

```
Boolean RetrieveHead { set; }
```

If set true (the default), then values from the head record are retrieved. If set false, values from the head record are omitted from any results, even if fields from the head record are added to the retrieval template.

Property: TdbRecord:RetrieveParts

Type: Boolean
Access: Write

Java

```
void setRetrieveParts(boolean mode)
```

.Net

```
Boolean RetrieveParts { set; }
```

If set true (the default), then values from available part records are retrieved. If set false, values from part records are omitted from any results, even if fields from the part record are added to the retrieval template.

Property: TdbRecord:PartId

Type: Integer
Access: Write

Java

```
void setPartId(int partId)
```

.Net

```
Int32 PartId { set; }
```

To restrict the retrieval to a particular part record (optionally including the head, unless the RetrieveHead property is set false), applications can set the PartId to the unique ID of the part to retrieve.

Retrieving SString fields

Note that the binary content of SString fields can be retrieved exactly as with any other field, i.e. by defining a TdbFieldTemplate for the field and adding that template to the retrieval record. In addition, however, applications can also request extra processing in the form of a rendition to be applied to the field prior to retrieval. In order to request a specific rendition, instead of creating a TdbFieldTemplate, applications should create an instance of a derived class called TdbRendition.

Class: TdbRendition

Derived from: TdbFieldTemplate
 Located in: data

This specialization of the field template allows the calling application to set the type of rendition that is to be produced during retrieval using the constructor:

Java

```
TdbRendition(String name, TdbRenditionType type)
```

.Net

```
TdbRendition(String name, TdbRenditionType type)
```

The rendition types currently supported via the TdbRenditionType enumeration are Default (i.e. binary, or no rendition), HTML, and Mime-Encoded HTML (MHTML file format, including all embedded graphics).

Processing results using TdbRecord

If the results of a retrieval operation are to be processed programmatically, or in some other way that doesn't allow for easy transformation from XML, the most useful way of interacting with the results is as a collection of TdbRecord instances.

To generate such a collection of records, use the simple form of the Get method on the record set and then the calling program can retrieve records from the result set using the Records collection.

Collection: TdbRecordSet:Records

Type: List of TdbRecord
 Access: Read

Java

```
List<TdbRecord> records()
```

.Net

```
IList Records { get; }
```

Each TdbRecord instance consists of a head component and, optionally, a list of part components. Each of these components is of type TdbComponent.

Class: TdbComponent

Derived from: Object
 Located in: data

To retrieve a reference to the head record, use the Head property:

Property: TdbRecord:Head

Type: TdbComponent
Access: Read

Java

```
TdbComponent getHead()
```

.Net

```
TdbComponent Head { get; }
```

Retrieve a reference to the record's head component.

Equally, to retrieve the collection of part record components, use the Parts property:

Collection: TdbRecord:Parts

Type: List of TdbComponent
Access: Read

Java

```
List<TdbComponent> parts()
```

.Net

```
ICollection Parts { get; }
```

Retrieve a list of available part record components.

To retrieve a specific component, use the GetComponent method:

Method: TdbRecord:GetComponent

Type: TdbComponent
Throws: TdbException

Java

```
TdbComponent GetComponent(int id)
```

.Net

```
TdbComponent GetComponent(int id)
```

Retrieve the specified component from the record, if it exists. If id is zero, the head component is returned; any value > 0 will return the appropriate part component, if available.

Each component offers access to the fields within the component using a variety of different access mechanisms, the simplest of which is to retrieve the collection of all fields.

Collection: TdbComponent:Fields

Type: List of TdbField
Access: Read

Java

```
Collection<TdbField> fields()
```

.Net

```
ICollection Fields { get; }
```

Retrieve the collection of fields. Each retrieved element will be of a type that implements the TdbField interface (in Java this is actually an abstract base class).

Interface: TdbField

Located in: data

To retrieve a specific field from the component, applications can use the method shown below.

Method: TdbComponent:GetField

Type: TdbField
Throws: TdbException

Java

```
TdbField getField(String fieldName)
```

.Net

```
TdbField GetField(String fieldName)
```

Each field offers methods and properties for retrieving field values. The two most important for this discussion are: OriginalValues, and Values. The former offers the content of the field, complete with any hit term markup specified, as retrieved from the server. The latter, in contrast, offers the content of the field as it exists at the current moment (more detail on updating data is given in section 7) and does not contain hit term markup.

Collection: TdbField:OriginalValues

Type: List of String
Access: Read

Java

```
List<String> originalValues()
```

.Net

```
ICollection OriginalValues { get; }
```

Retrieve the collection of field values, one String for each value (i.e. subfield or paragraph), complete with any hit term markup specified in the retrieval template.

Collection: TdbField:Values

Type: List of String
Access: Read

Java

```
List<String> values()
```

.Net

```
ICollection values { get; }
```

Retrieve the collection of values that reflects the field's current contents. Each String in the collection reflects a specific value from the field (i.e. subfield or paragraph). See section 7 of this guide for text field-specific circumstances under which this property is invalid.

Note that each field retrieved using either the GetField method or the Fields collection will be one of the following concrete types:

Class: TdbTextField

Derived from: TdbField
Located in: data

Class: TdbPhraseField

Derived from: TdbStructuredField
Located in: data

Class: TdbIntegerField

Derived from: TdbStructuredField
Located in: data

Class: TdbNumberField

Derived from: TdbStructuredField
Located in: data

Class: TdbDateField

Derived from: TdbStructuredField
Located in: data

Class: TdbTimeField

Derived from: TdbStructuredField
Located in: data

Class: TdbStringField

Derived from: TdbField
Located in: data

Note that in addition to the normal processing supported on fields, TdbStringField allows the application to request the rendered version of the field, using the Rendition property. This property is invalidated by any operation that updates the content of the field.

For an example of processing a record set after deserializing into a collection of TdbRecord objects, see the following examples.

Java

```
com.tietoenator.trip.jxp.examples.data.RecordSet  
com.tietoenator.trip.jxp.examples.data.View
```

.Net

```
Databases \ Data \ Explorer.vb
```

Processing results in XML

By using a different version of `TdbRecordSet`'s `Get` method, applications can retrieve the full XML document directly, rather than having the class library deserialize the document into a set of `TdbRecord` instances.

Method: `TdbRecordSet.Get`

Type: XML DOM Document
Throws: `TdbException`

Java

```
org.w3c.dom.Document get(TdbDataFormat format)
```

.Net

```
System.Xml.XmlDocument Get(TdbDataFormat format)
```

Retrieve the record set, returning the XML document sent from the server rather than deserializing the returned document into a sequence of records. The content of the record set is blank following this call.

Enumeration: `TdbDataFormat`

Located in: `data`

TdbDataFormat.RAW

Fields are represented within the returned document using `<FIELD>` elements. The `NAME` attribute of the `FIELD` element gives the field's name. For example:

```
<FIELD NAME="CHAPTER" TYPE="3">  
  <SUB ID="1">Chapter 1</SUB>  
</FIELD>
```

TdbDataFormat.ELEMENT

Fields are represented within the returned document using elements named for the field. Using the same field as shown above:

```
<CHAPTER>  
  <SUB ID="1">Chapter 1</SUB>  
</CHAPTER>
```

Structure of the XML response

Depending on the format requested (i.e. either `RAW` or `ELEMENT`), the content of each record will appear differently, although the framework within which record content appears will always be the same.

Each response will contain the following elements:

```
<FETCH_RES>
  [ <SUMMARY RECS="..." HITS="..." /> ]
  <RECORD ID="..." BASE="..." RANK="..." RAWRANK="...">
    ...
  </RECORD>
  <RECORD ... > ... </RECORD>
  ...
</FETCH_RES>
```

That is, all valid responses are bound in a root `FETCH_RES` element, and consist of a list of `<RECORD>` elements. If the response is generated by a query, i.e. if the `Query` property was set on the record set, the result will include a `SUMMARY` element detailing the total number of records and terms hit by the query.

Each `RECORD` element will specify at least four attributes, namely the record's ID, the database from which it was retrieved, the relevance rank (percentage), and the raw relevance rank (not necessarily a percentage, and not normalized) assigned to the record. Note that the `RANK` and `RAWRANK` attributes are really only useful following a non-Boolean query.

Record content formatted according to the `RAW` request type will appear with a `FIELD` element representing every field retrieved. For example:

```

<RECORD ID="98" BASE="ALICE" RANK="2">
  <FIELD NAME="CHAPTER" TYPE="3">
    <SUB ID="1">Pig and Pepper</SUB>
  </FIELD>
  <FIELD NAME="CHAPTNR" TYPE="9">
    <SUB ID="1">6</SUB>
  </FIELD>
  <FIELD NAME="SPEAKER" TYPE="3">
    <SUB ID="1">Cheshire Cat</SUB>
  </FIELD>
  <FIELD NAME="TXT" TYPE="2">
    <P> ... </P>
  </FIELD>
</RECORD>

```

Each field is named and identified with a field type, following which each subfield or paragraph is emitted.

This type of retrieval is useful in generic applications, where the application itself is not necessarily in control of what content is being retrieved, or where the style sheet that will be applied to the content is not specific to field name, but rather to field type.

Alternatively, in more controlled environments, the ELEMENT data format shows as in this example:

```

<RECORD ID="98" BASE="ALICE" RANK="2">
  <CHAPTER><SUB ID="1">Pig and Pepper</SUB></CHAPTER>
  <CHAPTNR><SUB ID="1">6</SUB></CHAPTNR>
  <SPEAKER><SUB ID="1">Cheshire Cat</SUB></SPEAKER>
  <TXT>
    <P> ... </P>
  </TXT>
</RECORD>

```

This type of retrieval is useful when the application is in complete control over what is being retrieved, and wishes to format its response using a field-specific style sheet.

Transforming the result XML

Regardless of the format of the request, the response will require transformation of some kind in order to be useful to end users. As the response from the request is a DOM Document, the application is at complete liberty as to the transformation that makes most sense to perform. This could include constructing a tree of objects (as the default deserialization to TdbRecord instances does), constructing a transport format such as JSON, constructing a representational format such as HTML or PDF, or indeed simply

constructing another XML document for further processing by other tiers or components of the application.

The following example shows how to turn a retrieved XML document from a TdbRecordSet request into a String using an XSLT style sheet.

Java

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

// Create the record set, set any retrieval parameters / template
TdbRecordSet rs = new TdbRecordSet(session);
...

// Make the request, return the result as an XML document
Document result = rs.get(TdbDataFormat.ELEMENT);

// Read the style sheet from somewhere, could be a file/URL/etc.
InputStream is = acquireStyleSheet();

// Create somewhere for our transformation to be written
ByteArrayOutputStream os = new ByteArrayOutputStream();

// Transform the result into a String
TransformerFactory
    .newInstance()
    .newTransformer(new StreamSource(is))
    .transform(new DOMSource(result), new
StreamResult(os));

// All done
String final_result = os.toString();
```

VB.Net

```
Imports System.IO
```

```
Imports System.Text
```

```
Imports System.Xml
```

```
Private xslt As String = "... your stylesheet ..."
```

```
' This procedure requests the record set from the server, then  
' transforms the resultant DOM Document using a stylesheet that  
' is created as a string (the "xslt" variable) and outputs the  
' the result of the transformation to a named file
```

```
Public Sub transformToFile(ByVal filename As String)
```

```
    ' Create the record set, set any retrieval parameters, etc.
```

```
    Dim rs As New TdbRecordSet(session)
```

```
    ...
```

```
    ' Retrieve the result document
```

```
    rs.Get(TdbDataFormat.ELEMENT)
```

```
    ' Create a transformation processor from our stylesheet
```

```
    Dim tx As New Xsl.XslTransform
```

```
    tx.Load(New XmlTextReader(New StringReader(xslt)), _  
            Nothing, Me.GetType().Assembly.Evidence)
```

```
    ' Create an output writer for the result file
```

```
    Dim xw As New XmlTextWriter(filename, Encoding.UTF8)
```

```
    ' Generate the transformation
```

```
    tx.Transform(doc.CreateNavigator(), Nothing, xw, Nothing)
```

```
    xw.Flush()
```

```
    xw.Close()
```

```
End Sub
```

Hit terms in the XML

Search hits are represented within the XML result using a <HIT> element containing the term in question. Even if two terms are coincident, each term will be separately contained within their own <HIT> element. For example, consider searching for the phrase “mad hatter”:

```
<SPEAKER>
  <SUB ID="1">March Hare</SUB>
  <SUB ID="2"><HIT>Mad</HIT> <HIT>Hatter</HIT></SUB>
</SPEAKER>
```

Summary

For examples of this kind of data transformation, see the following modules:

Java

```
com.tietoenator.trip.jxp.examples.data.Transform
com/tietoenator/trip/jxp/examples/data/html.xml
com/tietoenator/trip/jxp/examples/data/text.xml
```

.Net

```
Databases \ Data \ Transform.vb
```

6. TdbSearch

Rationale

The TRIPxpi client libraries TRIPnxp and TRIPjxp both contain numerous classes for search and retrieval, with TdbRecordSet and TdbCclCommand being the two most central ones.

So why introduce yet another search class, if the functionality is there already? To understand this, we must look at the designed behavior of the TdbCclCommand and TdbRecordSet classes. By understanding the kind of problems they solve, and more importantly the kind of problems they do not solve, we set the stage for the new classes described in this document.

TdbCclCommand

This class provides a front-end to the TRIP CCL command interpreter.

Pro:

- Any CCL command can be executed via this class.
- Provides access to output format reports via the TdbKernelWindow class.
- Provides access to term lists via the TdbTermList and TdbTermTree classes.

Con:

- A higher degree of network communication than with e.g. TdbRecordSet (still, lower than that of TRIPjtk and TRIPclient).
- Applications have to parse output format reports to get data for modification purposes.

TdbRecordSet

This class defines the notion of a set of records extracted from a database or search set, each record comprising a set of components, each component comprising a set of fields, each field comprising a set of values.

Pro:

- Network communication is kept to a minimum.
- Possible to fetch only the necessary data from found records.
- Good for stateless apps; does all in one go and cleans it up afterwards.
- Provides data in structured form via the classes TdbRecord, TdbField, etc.
- Possible to use search sets produced by TdbCclCommand.

Con:

- Impossible to reuse search sets produced by executing queries via the TdbRecordSet class.

- Requesting additional data from a search result involves executing the query again, which can result in a rather severe performance hit for large databases and/or complex queries.

Requirements

Stateless applications already seem to be covered rather nicely. However, we seem to lack something to make stateful application development as easy.

A new class for searching should have the following characteristics:

- Low degree of network I/O
- Reuse of pre-existing search sets.
- Provide data via output format reports or in structured form via the classes TdbRecord, TdbField, etc.
- Access term lists and trees.

In addition, we'd also like a few bells and whistles:

- Option for automatic cleanup operation (removal of search sets, etc) to make stateless use more practical.
- Iteration/enumeration of the records in a search set, without the application having to bother with explicitly fetching the next block of records from the server.

Of course, we can do all this already, by combining the use of TdbCclCommand and TdbRecordSet in an application. All the functionality is there. The crux is, of course, that it may not be obvious how to combine the necessary features so that all the items listed above are fulfilled.

So this is where TdbSearch comes into play. As stated, it contains nothing new; all the functionality it provides can be accomplished in an application using version 1.x of TRIPnxp or TRIPjxp. The idea with this new class is to provide access to pre-existing functionality with best practices for stateful applications in mind.

Creating a TdbSearch Instance

The TdbSearch Class

The TdbSearch class is available in the data package/namespace.

Class: TdbSearchSet

Derived from: TdbMessageProvider
Located in: data

Its behavior is a mix of the TdbCclCommand class and the TdbRecordSet class.

TdbSearch Constructor

There is only one constructor in the TdbSearch class.

Constructor: TdbSearch*Java*

TdbSearch(TdbSession session)

.Net

TdbSearch(TdbSession session)

The constructor creates a new, blank TdbSearch instance ready for use. This operation will cause network I/O. The following operations are done:

- Creates a kernel window of type HISTORY with default size.
- Creates a kernel window of type DISPLAY with default size.
- Creates a kernel window of type SHOW with 20 rows and 80 columns.
- Creates a kernel window of type SYSINFO with 20 rows and 80 columns.

So by creating a TdbSearch instance, you will already have caused four network transactions to be performed. This means that you should try to keep your TdbSearch instance alive for as long as you possibly could need it. Destroying it and recreating it whenever you need an instance of TdbSearch is not best practice.

Executing CCL Statements**Overview**

Using the TdbSearch instance for searching is similar to using the TdbCclCommand class. Utilizing an almost "classic" behavior, an executed CCL command that may result in a search set will only report back information about the search set. Data is not fetched at this point.

Method: TdbSearch:Execute

Type: void
Throws: TdbException

Java

void execute(String cclStatement)

.Net

void Execute(String cclStatement)

Although the TdbSearch class is mainly designed to support searching, the Execute method accepts any kind of CCL statement.

After a successful call to the Execute method, the state of the TdbSearch object will reflect the command just executed. To make sure what action is appropriate, check the CommandType property.

Property: TdbSearch.CommandType

Type: TdbCclCommandType
Access: Read

Java

```
TdbCclCommandType getCommandType()
```

.Net

```
TdbCclCommandType CommandType { get; }
```

Check the table below for what command types are associated with what type of CCL command.

Command type	Commands
HistoryUpdate	BASE, FIND, FUZZ
HistoryRenum	DELETE, RENUM
TermList	DISPLAY
TermTree	DISPLAY
Output	SHOW
Misc	(any other command)

Performing a Search

API Summary

A command like BASE, FIND or FUZZ generates a search set if successful. The following properties and methods are useful after a successful search operation:

- GetSearchSetById
- LastSearchSet
- SearchSetCount
- SearchSetNumbers
- SearchSets
- *.NET only*: indexer property
- *Java only*: getSearchSet

Search sets are represented by the class TdbSearchSet.

Class: TdbSearchSet

Derived from: TdbSessionObject
Located in: data

The TdbSearchSet object for the last search order is available via the LastSearchSet property. You can also access all search sets created using the current TdbSearch instance via the indexer property (.NET only), the getSearchSet method (Java only), or the GetSearchSetById method.

More about the TdbSearch class under the "Fetching Structured Data" topic on page 84.

Below follows brief description of the mentioned properties and methods.

Method: TdbSearch:GetSearchSetById

Type: TdbSearchSet
Throws: TdbException

Java

```
TdbSearchSet getSearchSetById(int searchId)
```

.Net

```
TdbSearchSet GetSearchSetById(int searchId)
```

Retrieve a TdbSearchSet object for the specified search id. If the search set does not exist, or if the search set is not created via the current TdbSearch instance, null is returned.

Property: TdbSearch:LastSearchSet

Type: TdbSearchSet
Access: Read

Java

```
TdbSearchSet getLastSearchSet()
```

.Net

```
TdbSearchSet LastSearchSet { get; }
```

This method returns a TdbSearchSet object for the last search conducted via the current TdbSearch instance.

The TdbSearchSet instance returned is to be regarded as volatile. A DELETE or RENUM command, for instance, may cause the search set to be deleted or renumbered. The application should therefore not keep any copies of TdbSearchSet instances, but instead access them when required via the properties on the TdbSearch class.

Property: TdbSearch:SearchSetCount

Type: int
Access: Read

Java

```
int getSearchSetCount()
```

.Net

```
int searchSetCount { get; }
```

This method returns the number of search sets associated with this TdbSearch object.

Property: TdbSearch:SearchSetNumbers

Type (java): int[]
 Type (.NET): IList<int>
 Access: Read

Java

```
int[] getSearchSetNumbers ()
```

.Net

```
IList<int> SearchSetNumbers { get; }
```

This method returns the numbers (or ids) of the search sets associated with this TdbSearch object. Only these numbers are valid as arguments to the GetSearchSetByld method.

Property: TdbSearch:SearchSets

Type (java): List<TdbSearchSet>
 Type (.NET): IList<TdbSearchSet>
 Access: Read

Java

```
List<TdbSearchSet> searchSets ()
```

.Net

```
IList<TdbSearchSet> SearchSets { get; }
```

This method returns a read-only list of the TdbSearchSet objects associated with the current TdbSearch instance.

Property: TdbSearch:Item

Type (.NET): TdbSearchSet
 Access: Read

Java

n/a

.Net

```
TdbSearchSet Item[int index] { get; }
```

This method is an indexer property and returns a TdbSearchSet instance. The index is the zero-based index of the search set to fetch, and is not to be confused with the id of the search. Valid index values range from 0 to one below the value returned by the SearchSetCount property.

Method: TdbSearch:getSearchSet

Type: TdbSearchSet
 Throws: TdbException

Java

```
TdbSearchSet getSearchSet(int index)
```

.Net

n/a

This method is used in TRIPjxp in place of an indexer property and returns a TdbSearchSet instance. The index is the zero-based index of the search set to fetch, and is not to be confused with the id of the search. Valid index values range from 0 to one below the value returned by the SearchSetCount property.

Search Example

The following simple example shows how to conduct a search and check its results.

C#

```
using TietoEnator.Trip.Nxp.Data;

void SearchTest (TdbSearch searchHandler)
{
    searchHandler.Execute("FIND MAD HATTER");
    // Here we really should check CommandType first,
    // but this IS a (very) simple example...

    TdbSearchSet searchSet = searchHandler.LastSearchSet;
    Console.WriteLine("{0} hits in {1} records",
        searchSet.HitCount,
        searchSet.RecordCount );
}
```

Fetching Structured Data

This topic deals with fetching data as instances of TdbRecord and its associated classes (TdbComponent, TdbField, etc). The idea is somewhat similar to that of TdbRecordSet, but does not require advance knowledge of how many records to retrieve.

What is similar to the behavior of TdbRecordSet is that before you retrieve anything, you will have to define a retrieval template. See chapter 5 (retrieving data from databases or search sets) in the *TRIPnXP & TRIPjXP Programmer's Guide* for more details about how to create retrieval templates.

Search Set Statistics

A search order executed via a TdbSearch instance gets a TdbSearchSet instance to represent it. It carries information about the search itself, such as the number of hits and records in the set, but also - and more importantly - it provides the means by which structured data (TdbRecord, TdbField, etc) is retrieved from the server.

When you have executed a search order and have retrieved the `TdbSearchSet` instance, you can examine the state of the search set using the following properties:

Property Name	Property Type	Description
DidYouMean	String	Returns a "did-you-mean" suggestion for the command just executed. Only applies to the FUZZ command. Is an empty string for everything else.
HitCount	Int32	The total number of hits in the search set.
Database	String	The name of the database or cluster associated with the search set. Available from version 7.2-1.
ParsedCommand	String	Retrieve a constructed version of the command with fully expanded tokens, suitable for display to an end user.
RecordCount	Int32	The number of records in the search set.
SearchId	Int32	The numeric ID of the search set.

Record Cache

Just like the `TdbRecordSet` does, the `TdbSearchSet` retrieves records in intervals. What differs is that the application does not have to specify any interval size, nor it have to explicitly fetch each "block" of records.

The `TdbSearchSet` class employs a cache of the last 10 blocks of 10 records fetched from the server. The cache size is configurable, but with the default setup, the cache can hold up to 100 records.

If the application requests a record from the `TdbSearchSet` that is not available in the cache, the appropriate block of 10 records will be automatically fetched from the server. If the cache is full, the block with the least recently accessed records will be removed from the cache.

BEST PRACTISE

Clear the cache using the `ClearCache` method before assigning a new retrieval template to a `TdbRecordSet`, or when you are done with your `TdbSearchSet` instance!

Record Retrieval

The `TdbSearchSet` class is, as mentioned, also used for data retrieval. Instances of `TdbRecord` can be retrieved for the records in the search set.

One way is to use the `getRecord` method in Java, and the `indexer` property in .NET.

Method: TdbSearchSet:getRecord

Type: TdbRecord
 Throws: TdbException

Java

```
TdbRecord getRecord(int index)
```

.Net

n/a

Returns a TdbRecord instance for a record in the current search set.

The index is the zero-based index of the search set to fetch, and is not to be confused with the record id. Valid index values range from 0 to one below the value returned by the RecordCount property.

Property: TdbSearch:Item

Type (.NET): TdbRecord
 Access: Read

Java

n/a

.Net

```
TdbRecord Item[int index] { get; }
```

This method is an indexer property and returns a TdbRecord instance for a record in the current search set.

The index is the zero-based index of the search set to fetch, and is not to be confused with the record id. Valid index values range from 0 to one below the value returned by the RecordCount property.

If the requested record is in the cache, it will be returned from there. Otherwise, the appropriate block of 10 records will be fetched from the server, cached, and the requested record returned to the caller. If the cache has reached its size limit, the least recently used block of records will be removed from the cache to make room for the new one.

TdbSearchSet in for-each loops

The TRIPjxp version of TdbSearchSet implements the Iterable interface and the TRIPnpx version implements the IEnumerable interface. This means that the TdbSearchSet instance can be used in a “for-each loop” in both the Java and the .NET environment.

C#

```
foreach ( TdbRecord rec in searchSet )
{
    // Use the record instance here...
}
```

Automatic Retrieval Templates

The TdbSearch class can create automatic retrieval templates for the search sets generated by this class. The templates created consist of all fields in the database queried. This is default behavior.

Property: TdbSearch:AutomaticRetrievalTemplate

Type: boolean
Access: Read, Write

Java

```
boolean getAutomaticRetrievalTemplate()  
void setAutomaticRetrievalTemplate(boolean enable)
```

.Net

```
bool AutomaticRetrievalTemplate { get; set; }
```

If you choose to use automatic retrieval templates, you don't have to do anything about retrieval templates at all. However, you are strongly recommended to disable this behavior (setting this property to false) if either of the following is true.

- You query a cluster. If this property remains set to true, the retrieval template created by the TdbSearch class will not include any field templates.
- You do not intend to use all fields in the records retrieved.

Assigning a Custom Retrieval Template

You will have to use your own, custom retrieval template if you are using a database cluster, or if you only wish to retrieve values from some of the fields in a database.

BEST PRACTISE

Even if an automatic retrieval template happens to be a valid choice, you should seriously consider always using custom retrieval templates. Being explicit gives you control over exactly what you are retrieving!

The following example shows how to use a custom retrieval template with a search set.

C#

```

private TdbSession session;
private TdbDatabaseDesign db;
private TdbRecord template;
private TdbSearch search;

int main(String[] args)
{
    OneTimeSetup();
    SearchTest2();
    return 0;
}

void OneTimeSetup()
{
    // Log on to a TRIP server
    session = new TdbTripNetSession("localhost",23457);
    session.Login("myuser","mypass");

    // Fetch the DB designs only once per session.
    TdbDatabaseDesign db = new TdbDatabaseDesign(session)
    db.Get("ALICE");

    // Templates you might as well create now too.
    template = new TdbRecord(session,db,false);
    template.AddToTemplate("CHAPTER");
    template.AddToTemplate("TXT");

    // Not strictly required here, but a good idea to keep
    // the search object alive for as long as they're needed.
    search = new TdbSearch(session);
    search.AutomaticRetrievalTemplates = false;

    // Open database. Only relevant here if we're only using
    // a single database or cluster.
    search.Execute("BASE " + db.Name);
    search.Clear();
}

void SearchTest2()
{
    // Execute a search order
    search.Execute("FIND JABBERWOCKY");

    // Get the search set and assign the template to it.
    TdbSearchSet searchSet = searchHandler.LastSearchSet;
    searchSet.RetrievalTemplate = template;

    // Fetch the results from the server
    foreach (TdbRecord rec in searchSet)
    {
        Console.WriteLine(rec.Head["CHAPTER"].ToString());
        Console.WriteLine(rec.Head["TXT"].ToString());
    }

    // IMPORTANT: Clear the search object when you don't
    // need the search sets associated with it anymore.
    search.Clear();
}

```


Using Output Formats

If you wish to retrieve the result of a search using an output format, you can. Just execute the SHOW order via the same TdbSearch object you used to do the search. If successful and the CommandType is Output, the property AffectedWindow contains the TdbKernelWindow object with which you can fetch the output format report.

Example (error handling omitted):

```
C#
search.Execute("FIND MAD HATTER");
TdbSearchSet searchSet = search.LastSearchSet;
search.Execute("SHOW FORMAT=SHORT S=" +
               searchSet.SearchId.ToString());
output = search.AffectedWindow;
do
{
    # Print the lines in the buffer.
    Console.WriteLine(output.ToString());

    # Scroll down the window one page.
    output.ScrollDown();
}
while ( output.IsAtBottom == false );
```

7. Retrieving data from CONTROL

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

The purpose of the classes described in this section is to provide a simple and consistent mechanism for retrieving lists of information as would normally be available via CCL commands such as “show base list” or “show base access”, etc.

Retrieving information via these list classes removes any requirement for understanding the format of CONTROL, the format of the reports used to show information from CONTROL (as generated by the CCL commands referenced above), etc.

The base class of all such list classes is:

Class: TdbControlObjectList

```
Derived from:    TdbMessageProvider
Located in:      control
```

The TdbControlObjectList class is an enumerable list, natively supporting the “for-each” pattern. Each element of the list is of type TdbControlObject, as described below.

Control objects

A TdbControlObject instance is a reference to a particular TRIP entity, for example a user or a database. Lists of these Control object references are retrieved using a derivation of the list class TdbControlObjectList. Each Control object reference contains information from the Control database that uniquely references the specific TRIP entity to which it refers.

For example, a database entity within TRIP would be represented by a TdbControlObject with the following standard properties:

```
Name:           <database name>
Owner:           <name of the FM of the database>
CreateDate:      <date on which the design was created>
CreateTime:      <time at which the design was created>
ModifyDate:      <date on which the design was last modified>
ModifyTime:      <time at which the design was last modified>
Type:            Database (or thesaurus)
Comment:         <any description assigned to the database>
```

In addition, due to the object referenced being a database, the object will also have extended, or custom, properties such as:

```
RecordCount:     <number of records in the database>
ExtendedType:     <User, System, Demo, etc.>
xml?:            <Yes / No>
```

Similarly, every entity type within the TRIP system can be represented using a TdbControlObject. This class provides a large number of properties and methods for

interrogating the Control reference, allowing the programmer to determine exactly what it is, but equally the programmer can simply pass a `TdbControlObject` into the constructor of many different entity manipulation classes, such as `TdbDatabaseDesign`, allowing for completely generic application construction.

Class: TdbControlObject

Derived from: Object
Located in: root

Each instance of the `TdbControlObject` class contains a standard set of properties shared by all data types and an extensible set of custom properties that are specific to the type of data retrieved.

Creating and using Control object lists

In order to generate a list of information from the Control database, simply construct an object of the desired class and then iterate over the object collection, as shown in the following example.

Java

```
// The constructor actually makes the request of the server
TdbControlObjectList list = new TdbDatabaseList(session);

// At this point, the list is populated, so we can simply
// iterate over its content
for( TdbControlObject db : list )
{
    // ... do whatever is required with the object ...
}
```

VB.Net

```
' The constructor actually makes the request of the server
Dim list As New TdbDatabaseList(session)

' At this point, the list is populated, so we can simply
' iterate over its content
For Each db As TdbControlObject In list
    ' ... do whatever is required with the object ...
Next
```

In general there is at least one, and sometimes several, list class that pertains to each type of TRIP entity.

Class: TdbClassificationSchemesList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of available classification schemes. Classification schemes are specialized databases optionally accompanied by one or more data files specific to the classification algorithm in use by the scheme.

Class: TdbDatabaseAccessList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of users and/or groups who have some level of access to the database or cluster provided to the constructor. For each object in the list, the calling application can retrieve the level of access granted using the custom properties ReadAccess and WriteAccess, both of which return a value from the TdbAccessRights enumeration.

Class: TdbDatabaseFieldList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of fields from the design for a specific database or thesaurus. Various constructors are available to allow the calling application to specify field types or component memberships that are of interest. Note that the resulting Control objects contain only field names and no other field attributes. If the calling application requires field attributes, it must retrieve the database design (see section 10).

Class: TdbDatabaseList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of databases, thesauri and clusters to which the calling user has some level of access. An additional constructor is available to allow the calling application to restrict the list to a particular extended type of database. The retrieved objects can be tested for type using the custom properties IsDatabase, IsCluster, IsThesaurus(), IsDemoDatabase(), IsSystemDatabase() and IsUserDatabase().

Class: TdbEntryFormList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of TRIPclassic data entry forms defined for the database or thesaurus provided to the constructor.

Class: TdbFileManagerList

Derived from: TdbUserList
Located in: control

Retrieve a list of users who have file manager privilege. In order to use this class, the calling user must have some level of management privilege, either FM, UM or SM.

Class: TdbGroupAccessList

Derived from: TdbControlObjectList
Located in: control

Retrieve the access rights for a named group, i.e. all databases, thesauri and clusters to which the group has some level of access, and what the level of access is. Calling applications can retrieve the level of access from the retrieved objects using the properties ReadAccess and WriteAccess, both of which return a value from the TdbAccessRights enumeration.

Class: TdbGroupList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of groups owned by the calling user. Alternative constructors allow the list to pertain to a specific user, which must be a user owned by the calling user, and/or to include or exclude the PUBLIC group.

Class: TdbGroupMemberList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of group members.

Class: TdbManagerList

Derived from: TdbUserList
Located in: control

Retrieve a list of users with any kind of management privilege. In order to use this class, the calling user must themselves have management privilege of some kind.

Class: TdbOutputFormatList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of output formats defined for the database or thesaurus named in the constructor.

Class: TdbOwnedDatabaseList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of databases, thesauri or clusters owned by the calling user.

Class: TdbProcedureList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of procedures or macros owned by a specific user or group.

Class: TdbPublicAccessList

Derived from: TdbGroupAccessList
Located in: control

Retrieve a list of databases, thesauri or clusters to which the special PUBLIC group has been granted some level of access. Calling applications can retrieve the level of access from the retrieved objects using the properties ReadAccess and WriteAccess, both of which return a value from the TdbAccessRights enumeration.

Class: TdbPublicMemberList

Derived from: TdbGroupMemberList
Located in: control

Retrieve a list of all users on the system, i.e. all members of the special PUBLIC user group.

Class: TdbSearchFormList

Derived from: TdbControlObjectList
Located in: control

Retrieve a list of all TRIPclassic search forms.

Class: TdbUserAccessList

Derived from: TdbControlObjectList
Located in: control

Retrieve the access rights for a named user, i.e. all databases, thesauri and clusters to which the user has some level of access, and what the level of access is. Calling applications can retrieve the level of access from the retrieved objects using the properties ReadAccess and WriteAccess, both of which return a value from the TdbAccessRights enumeration.

Class: TdbUserList

Derived from: TdbControlObjectList
Located in: control

Retrieve the list of users owned by the calling user, optionally including the calling user's object. Alternative constructors support retrieving users owned by other user managers, or retrieving users of a certain management privilege level.

Class: TdbUserManagerList

Derived from: TdbUserList
Located in: control

Retrieve a list of users who hold the user manager privilege. In order to use this class, the calling user must themselves have management privilege of some kind.

There are several examples of retrieving information from the Control database, which can be found at the following locations.

Java

```
com.tietoenator.trip.jxp.examples.control.DatabaseList  
com.tietoenator.trip.jxp.examples.control.ListGenerator
```

.Net

```
Databases \ DatabasesNode.vb  
Forms \ *.vb  
Schemes \ SchemesNode.vb  
Users \ *.vb
```

Transforming Control object lists

The content of any given Control object list is a set (potentially ordered) of TdbControlObject elements, each of which is potentially specialized according to the derived type of the list in question. In certain circumstances, however, it is useful to be able to transform those Control objects into objects of other types, for example to place within web controls using frameworks such as JSF.

For this purpose, all Control object lists support a method that provides for arbitrary transformation of each element within the list to a new object type, the result of the operation being a new list of such new object types.

To support such transformation, the calling application must provide an instance of a class that implements the interface TdbTransformer. This simple interface defines a single method that transforms a Control object into an arbitrary object (note that this is a strongly-typed operation under Java only).

Database List Performance Considerations

When using the TdbDatabaseList to fetch a list of databases and clusters from TRIPsystem, the presence of clusters may have a noticeably negative performance impact on this operation. This is mainly due to the fact that this list will also include record counts by default. Record counts are expensive to produce for clusters, hence the performance hit. If you don't need the record counts, you should consider using the NoCounts database list type, which is much faster but with the caveat that you don't get any record counts.

For example:

```
TdbControlObjectList list = new TdbDatabaseList(session,  
        TdbExtendedDatabaseType.Any,  
        TdbDatabaseListType.NoCounts);
```

Interface: TdbTransformer

Derived from: Object
 Located in: <root>

Method: TdbTransformer:Transform

Type: Specialized object
 Throws: N/A

Java

```
public interface TdbTransformer<E> {
    public E transform(Object o);
};
```

.Net

```
public interface TdbTransformer {
    object Transform(Object o);
};
```

Method: TdbControlObjectList:Transform

Type: List of objects
 Throws: N/A

Java

```
class mytx implements TdbTransformer<String>
{
    public String transform(Object o)
    {
        return ((TdbControlObject)o).getName();
    }
};
```

```
TdbControlObjectList l = new TdbDatabaseList(session);
List<String> names = l.transform(new mytx());
```

C#

```
class mytx : TdbTransformer
{
    public string Transform(object o) {
        return ((TdbControlObject)o).Name;
    }
};
```

```
TdbControlObjectList l = new TdbDatabaseList(session);
IList names = l.Transform(new mytx());
```


8. Updating databases

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Updating a database is performed using the same classes as were used for retrieving data from databases, i.e. the TdbRecord and TdbRecordSet classes. The former is used when updating, deleting or inserting a single record, the latter when projecting a common update onto a set of records, when deleting a set of records, or when inserting a set of records at once.

Class: TdbRecord

Derived from: TdbMessageProvider
Located in: data

Class: TdbRecordSet

Derived from: TdbMessageProvider
Located in: data

In either case, the content of one or more TdbRecord instances must be established or modified locally, after which various type of commit operations can be performed.

Creating or retrieving single records

To create a new TdbRecord instance for data update purposes, associate the new TdbRecord with the database design for the database that is to be affected.

Constructor: TdbRecord

```
TdbRecord(TdbSession session,
          TdbDatabaseDesign design,
          Boolean createTemplate)
```

```
TdbRecord(TdbSession session,
          String name,
          Boolean createTemplate)
```

```
TdbRecord(TdbSession session,
          TdbControlObject obj,
          Boolean createTemplate)
```

Create a new TdbRecord instance associated with the database whose name or design is provided. The constructor reads the design (fetching it from the server if necessary) and establishes all required relationships between the record and the design, such as the existence and identity of key fields, etc.

If the “createTemplate” flag is set true, a retrieval template containing all fields within the associated design is automatically created, so that any following Get operation will retrieve records in their entirety from the server.

If this flag is set false, the application must explicitly add fields to the retrieval template before fetching records for modification.

To retrieve a specific record from the database set the record's RecordId or RecordName property and request the record from the server using the Get method as shown the following example.

Java

```
// Retrieve the design for the database we're going to
// work with
TdbDatabaseDesign db = new TdbDatabaseDesign(session);
db.get("alice");

// Create a new record instance, retrieve a record from
// the database
TdbRecord record = new TdbRecord(session, db, true)
record.setRecordId(32);
record.get();

// Modify the record's content as required
...

// Store the record back to the server
record.commit();
```

VB.Net

```
' Retrieve the design for the database we're going to
' work with
Dim db As New TdbDatabaseDesign(session)
db.Get("alice")

' Create a new record instance, retrieve a record from
' the database
Dim record As New TdbRecord(session, db, true)
record.RecordId = 32
record.Get()

' Modify the record's content as required
...

' Store the record back to the server
record.Commit()
```

Note that it is good practice to use a database design object rather than the database's name if the application will be creating more than one TdbRecord instance. When provided with a name, the constructor must fetch the database design from the server every time it is needed.

When creating a TdbRecord instance for an insert operation, the database design should still be provided to the constructor, although the retrieval template can be left blank, as shown in the following example.

Java

```
// Retrieve the design for the database we're going to work with
TdbDatabaseDesign db = new TdbDatabaseDesign(session);
db.get("alice");
```

```
// Create a new record instance
TdbRecord record = new TdbRecord(session, db, false)
```

```
// Establish the new record's content as required
...
```

```
// Store the record to the server as an insert
record.commit();
```

VB.Net

```
' Retrieve the design for the database we're going to work with
Dim db As New TdbDatabaseDesign(session)
db.Get("alice")
```

```
' Create a new record instance
Dim record As New TdbRecord(session, db, true)
```

```
' Establish the new record's content as required
...
```

```
' Store the record to the server as an insert
record.Commit()
```

In essence, the only difference between an insert operation and an update operation is whether the record was retrieved from the server before being committed or not. If the record has been affected by a Get operation, then any Commit operation will update the record, whilst in the absence of a preceding Get operation, the Commit operation will insert the record.

Method: TdbRecord:Commit

Type: void
Throws: TdbException

Java

```
void commit()
```

.Net

```
void Commit()
```

Commit the record to the database on the server, using either update or insert semantics depending upon whether the record was first retrieved from the database or not.

Modifying or establishing the content of a TdbRecord

Whether the application is creating new records or updating existing records, the method for modifying the TdbRecord instance is the same, although obviously the operations that occur prior to modification are different.

In order to modify a TdbRecord, first retrieve a reference to the appropriate component of the record, using the Head property, the Parts collection, or the GetComponent method.

Property: TdbRecord:Head

Type: TdbComponent
Access: Read

Java

```
TdbComponent getHead()
```

.Net

```
TdbComponent Head { get; }
```

Retrieve a reference to the head component of the record.

Collection: TdbRecord:Parts

Type: List of TdbComponent
Access: Read

Java

```
List<TdbComponent> parts()
```

.Net

```
ICollection Parts { get; }
```

Retrieve a list of part record components currently defined in the record.

Method: TdbRecord:GetComponent

Type: TdbComponent
Throws: TdbException

Java

```
TdbComponent getComponent(int index)
TdbComponent getComponent(String name)
```

.Net

```
TdbComponent GetComponent(int index)
TdbComponent GetComponent(String name)
```

Retrieve the record component at the specified index, or with the specified name. The zero'th component is the head record, whilst components numbered from 1 upwards are part records. Using the named version will work only if the TdbRecord instance is associated with a database design that uses a part record name field.

New part records can be created using the AppendComponent method.

Method: TdbRecord:AppendComponent

Type: TdbComponent
Throws: TdbException

Java

```
TdbComponent appendComponent()
```

.Net

```
TdbComponent AppendComponent()
```

Add a new part record to the end of the current record's part vector. This method cannot be used to create a new head record, obviously. If the part record structure requires a part record name, use the new component's Name property to establish this value. The new component's part number can be retrieved using the component's Id property.

Existing part records can be deleted using the DeleteComponent method.

Method: TdbRecord:DeleteComponent

Type: void
Throws: TdbException

Java

```
void deleteComponent(int id)
void deleteComponent(String name)
```

.Net

```
void DeleteComponent(Int32 id)
void DeleteComponent(String name)
```

Delete the named or identified part record from the record structure. The named version is usable only when the record is associated with a database design that

uses a part record name field, and only if the part record's name was established either by retrieval from the server or by the application having established the part name beforehand.

From the component, individual fields are retrieved using the class indexer (.Net only), the Fields collection, or the GetField method.

Property: TdbComponent:Item (.Net only)

Type: TdbField
Access: Read, Write

C#

```
TdbField this[String name] { get; set; }
```

VB.Net

```
Public Property Item(ByVal name As String) As TdbField
```

Retrieve a reference to the named field, if any (returns null/nothing if not found).

Collection: TdbComponent:Fields

Type: Collection of TdbField
Access: Read

Java

```
Collection<TdbField> fields()
```

.Net

```
ICollection Fields { get; }
```

Retrieve the collection of fields defined for the component to date. The collection is not ordered in any specific way. The collection is live, however, supporting update to individual fields within the collection, although the collection cannot be extended.

Method: TdbComponent:GetField

Type: TdbField
Throws: TdbException

Java

```
TdbField getField(String name)
```

.Net

```
TdbField GetField(String name)
```

Retrieve a reference to the named field, if found. The reference returned is live and can be updated freely.

New fields can be created using the CreateField method.

Method: TdbComponent:CreateField

Type: TdbField
Throws: TdbException

Java

```
TdbField createField(TdbFieldDesign field)
TdbField createField(String name, TdbFieldType type)
```

.Net

```
TdbField CreateField(TdbFieldDesign field)
TdbField CreateField(String name, TdbFieldType type)
```

Create a new field instance within the component for a field of the defined design, or with the given name and the given field type. This method will unconditionally overwrite any existing field with the same name within the component.

The TdbField instance returned will be of a concrete type that is appropriate to the field type requested, e.g. TdbTextField, TdbPhraseField, etc.

Existing fields can be deleted using the DeleteField method.

Method: TdbComponent:DeleteField

Type: void
Throws: N/A

Java

```
void deleteField(String name)
```

.Net

```
void DeleteField(String name)
```

Delete the named field from the component. If the field is not found the operation is legal but has no effect. If the field is found, the field is maintained as a member of the component, but has a blank value—this ensure that when the record is committed to the server, the field's value will be deleted.

Working with structured field types

Having either created or retrieved the field required, the field's current values can be retrieved using the Values collection or the GetValue method.

Collection: TdbField:values

Type: List of String
Access: Read

Java

```
List<String> values()
```

.Net

```
ICollection values { get; }
```

Retrieve the list of values currently stored within the field. In contrast to the OriginalValues collection described in section 5, the Values collection does not

contain hit term markup and always reflects any modifications that have been made to the field locally.

Method: TdbField.GetValue

Type: String
Throws: TdbException

Java

```
String getValue(int index)
```

.Net

```
String GetValue(Int32 index)
```

Retrieve the field value at the specified offset, i.e. the value of the specified subfield. This method throws an exception if the specified subfield index is out of bounds for the current content of the field.

The values that the field holds can be modified using the SetValue and AppendValue methods.

Method: TdbField.SetValue

Type: void
Throws: TdbException

Java

```
void setValue(int index, String value)
```

.Net

```
Void SetValue(Int32 index, String value)
```

Establish the value of the subfield at the defined index, throwing an exception if the index is out of bounds of the current field content. Values intended for fields of a non-textual nature, e.g. numbers, are validated prior to updating with the value given.

Method: TdbField.AppendValue

Type: void
Throws: TdbException

Java

```
void appendValue(String value)
```

.Net

```
void AppendValue(String value)
```

Append the defined value to the current content of the field, as a new subfield.

The following example shows a simple insert being performed to the Alice database.

Java

```
TdbDatabaseDesign db;
TdbComponent      head;
TdbRecord          record;
TdbField           field;

// Retrieve database design, create new record
db = new TdbDatabaseDesign(session);
db.get("alice");
record = new TdbRecord(session, db, false);
head = record.getHead();

// Create field values
field = head.createField(db.getField("Chaptnr"));
field.appendValue("99");

field = head.createField(db.getField("Chapter"));
field.appendValue("Chapter title");

// Store new record to database
record.commit();
```

VB.Net

```
' Retrieve database design, create new record
Dim db As New TdbDatabaseDesign(session)
db.Get("alice")
Dim record As New TdbRecord(session, db, false)
Dim head As TdbComponent = record.Head

' Create field values
Dim field As head.CreateField(db("chaptnr"))
field.AppendValue("99")

field = head.CreateField(db("chapter"))
field.AppendValue("Chapter title")
```

```
' store new record to database
record.Commit()
```

Working with TEXT fields

Although the `TdbTextField` class supports the value-based operations described in the previous section (with certain restrictions), TEXT fields tend to require slightly differently operating procedures than other field types, as dictated by the kernel requirements of the field type itself.

Whilst it is simple for an application to divide the values in a PHrase field into subfields, for example, simply by delimiting input on newlines, TEXT field values can be significantly different from one database to another or one application to another, and so expecting application developers to deal with this complexity is unreasonable.

Therefore, in addition to the `SetValue` and `AppendValue` methods, TEXT fields can be modified using the `SetText` and `AppendText` methods. For the .Net platform only, there is an additional property, the `Text` property, which is intended for use with the `Text` property of edit controls and similar UI widgets.

Method: `TdbTextField.SetText`

Type: void
Throws: N/A

Java

```
void setText(String value)
```

.Net

```
void SetText(String value)
```

Establish the content of the TEXT field. Any pre-existing content will be removed by this method.

Applications must be careful to provide the text values exactly as the user enters them, maintaining all white space and newline equivalents, as these characters are used by the server to parse the provided text into paragraphs and sentences.

Method: `TdbTextField.AppendText`

Type: void
Throws: N/A

Java

```
void appendText(String value)
```

.Net

```
void AppendText(String value)
```

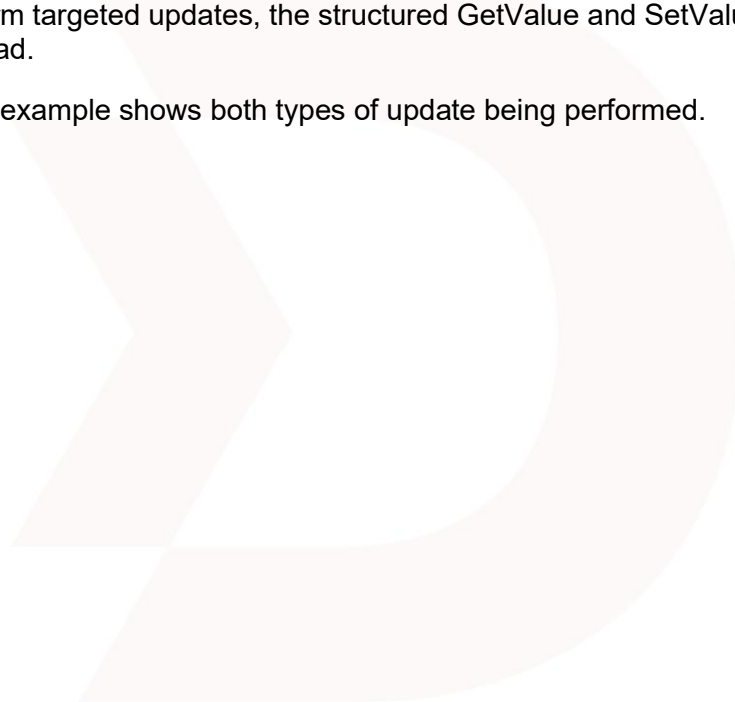
Establish the value of the text field, or if the text field already has text content, as defined by previous calls to either `SetText` or `AppendText`, append the provided value to the current content.

Applications must be careful to provide the text values exactly as the user enters them, maintaining all white space and newline equivalents, as these characters are used by the server to parse the provided text into paragraphs and sentences.

Immediately following retrieval from the server using the Get method, and before any updates have been performed on the field, a TExt field will in fact be divided neatly into paragraphs that can uniquely be accessed and updated using the GetValue and SetValue methods. This can be very useful when performing targeted updates, of course. However, as soon as the application invokes either SetText or AppendText on the field, the value-based methods become invalid and will throw an exception due to the fact that the SetText/AppendText methods intentionally do not attempt to divide the provided text into paragraphs.

Thus, it is important for application developers to understand their end user's requirements in terms of interacting with TExt fields. If the interaction is going to be UI-centric, focusing on the user editing the content of a TExt field, then the application should write the field's value using the SetText and AppendText methods. If, however, the application is to be used to perform targeted updates, the structured GetValue and SetValue methods should be used instead.

The following example shows both types of update being performed.



Java

```
TdbDatabaseDesign db;

TdbComponent      head;
TdbTextField      field;
TdbRecord          record;

// Retrieve database design and record from database
db = new TdbDatabaseDesign(session);
db.get("alice");
record = new TdbRecord(session, db, true);
record.setRecordId(32);
record.get();
head = record.getHead();

// Use structured update to modify paragraph 1 of
// the TXT field
field = (TdbTextField)head.getField("txt");
if( field != null )
    field.setValue(1, "This replaces para 1 of the field");

// Use unstructured update to establish new TXT2 field content
field = (TdbTextField)head.createField(db.getField("txt2"));
field.setText
    ("Initial field value.\nSome more of the field.");
field.appendText("More text for the field.\n\n");

// Store the modified record to the server
record.commit();
```

VB.Net

```
' Retrieve database design
Dim db As New TdbDatabaseDesign(session)
db.Get("alice")

' Retrieve record to be modified
Dim record As New TdbRecord(session, db, true)
record.RecordId = 32
record.Get()

Dim head As TdbComponent = record.Head
Dim field As TdbTextField

' Use structured update to modify paragraph 1 of the TXT field
field = head("txt")
If Not field Is Nothing Then
    field.SetValue(1, "This replaces para 1 of the field")
End If

' Use unstructured update to establish new TXT2 field content
field = head.CreateField(db("txt2"))
field.SetText("Initial field value.\nSome more of the field.")
field.AppendText("More text for the field.\n\n")

' Store the modified record to the server
record.Commit()
```

Working with SString fields

TRIP's binary large object field type, SString, is represented within the class library using the TdbStringField class. This is a very restricted implementation of the TdbField interface, as this field type does not support getting and/or setting individual values within the field, just the entire value of the field in one operation.

To support this interaction, the class defines the Blob property.

Property: TdbStringField:Blob

Type: Array of bytes
Access: Read, Write

Java

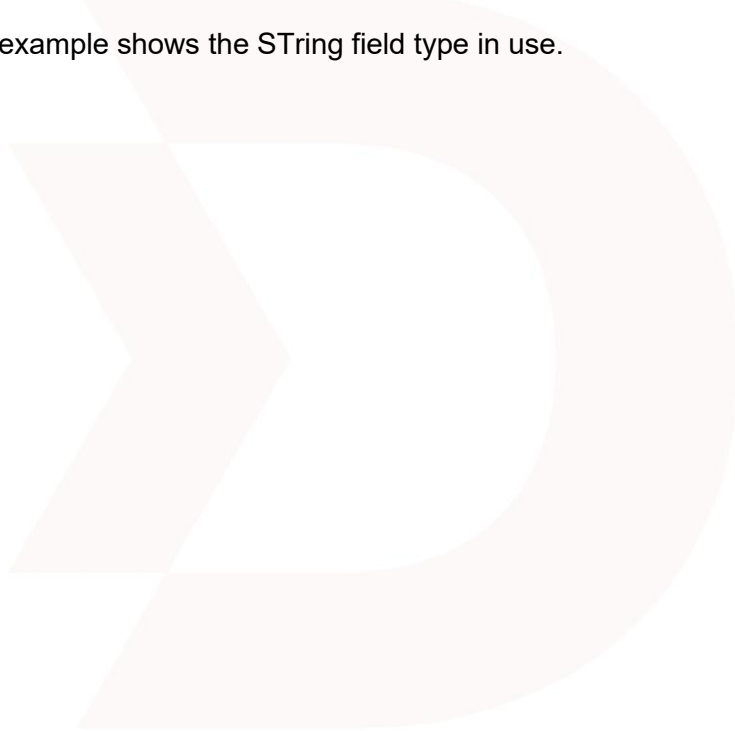
```
byte[] getBlob()  
void setBlob(byte[] value)
```

.Net

```
Byte[] Blob { get; set; }
```

Retrieve or establish the value of the field. Note that the byte array used for all interactions is “live” in order to avoid unnecessary memory usage by copying values to/from the field’s content. Programmers must be careful when using this array, therefore, so as to avoid corrupting field values inadvertently.

The following example shows the SString field type in use.



Java

```
TdbDatabaseDesign db;
TdbStringField    field;
TdbRecord         record;

// Load the database design and the appropriate record from
// the DB
db = new TdbDatabaseDesign(session)
db.get("some_db");
record = new TdbRecord(session, db, true);
record.setRecordName("My Unique Key");
record.get();

// Get the existing value of the string field
field = (TdbStringField)record.getHead().getField("my_blob");
byte[] blob = field.getBlob();

// Launch an app to deal with the blob, generate a new one
myApp.launch(blob);
field.setBlob(myApp.getNewValue());

// Store record updates
record.commit();
```

VB.Net

```
' Load the database design and the appropriate record
' from the DB
Dim db As New TdbDatabaseDesign(session)
db.Get("some_db")
Dim record As New TdbRecord(session, db, true)
record.Name = "My Unique Key"
record.Get()

' Get the existing value of the string field
Dim field As TdbStringField = record.Head("my_blob")
```



```
Dim blob() As Byte = field.Blob

' Launch an app to deal with the blob, generate a new one
myApp.launch(blob)
field.Blob = myApp.NewValue

' Store record updates
record.Commit()
```

Deleting single records

In order to delete a single record from a database or thesaurus, create a `TdbRecord` instance as usual, identify the record that is to be deleted, using either the `RecordId` or `RecordName` property, and then invoke the `Delete` method to remove the record from the database.

Method: `TdbRecord.Delete`

Type: `void`
Throws: `TdbException`

Java

```
void delete()
```

.Net

```
void Delete()
```

Deletes the current record, as identified by the `RecordId` or `RecordName` property, from the database with which the record is associated, as defined by the `Database` property or the constructor.

The following example illustrates how to call this method.

Java

```
TdbRecord record = new TdbRecord(session);

record.setDatabase("My_Database");
record.setRecordName("My unique key");
record.delete();
```

VB.Net

```
Dim record As New TdbRecord(session)

record.Database = "My_Database"
record.RecordName = "My unique key"
record.Delete()
```

In this particular case, where there's no need to create a field-specific association between the record and the database, we can construct the TdbRecord without having loaded the database design, and simply reference the database by name.

Affecting multiple records with one request

In addition to interacting with single records, using the TdbRecord class, applications can interact with multiple records at once using the TdbRecordSet class. The semantics of these operations vary depending on the type of operation, as described below.

- Multiple insert adds a set of records to a single database with one request.
- Multiple update performs the same update operation on a set of records.
- Multiple delete removes a set of records from a database with a single request.

Multiple insert

In application scenarios involving mass inserts to a database, where granular insertion validation isn't necessarily an issue, for example offline batch loading, it can be very useful to be able to state an insertion request using multiple records at once.

To do this, the TdbRecordSet class offers an Insert method that takes as argument a collection of TdbRecord instances. Records within the collection do not themselves need to be established as rigorously as when performing a singular operation, as the application must instead define the context of the TdbRecordSet in order for the operation to complete.

Method: TdbRecordSet:Insert

Type: void
Throws: TdbException

Java

```
void Insert(Collection<TdbRecord> records)
```

.Net

```
void Insert(ICollection records)
```

Insert the collection of records to the current database. The TdbRecordSet on which this method is invoked must be prepared by setting the Database property to identify the database into which the inserts should be performed. If the database uses a record name field, each record in the collection must have their record name established by using the Name property of the TdbRecord instance.

Following the successful invocation of this method, the property AffectedRecords will reflect the number of records inserted during the operation.

Multiple update

The purpose of multiple update is to allow an application to project a single update operation onto a set of records identified by record range or query result. For example, to change the value of a particular field in all records in which that field currently has an existing incorrect value.

This facility is less granular than the global update capabilities offered by CCL, as it does not allow for individual words or phrases to be updated based on query results, but if the update that is required can be defined in terms of rigorous database structure, then this facility provides a quick and online means of accomplishing the purpose.

To perform a multiple update, prepare the record set by setting some combination of the Query, Database, From, To, Id, or Name properties and then invoke the Update method.

Method: TdbRecordSet:Update

Type: void
Throws: TdbException

Java

```
void update(TdbRecord values)
```

.Net

```
void Update(TdbRecord values)
```

Perform the update operation identified by the provided TdbRecord on every record covered by the record set's definition. The record set can be prepared either using a database name or using a query statement, thus allowing records from multiple databases to be updated.

Multiple delete

To delete a range of records from a database, or a range of records covered by a query result, simply prepare a TdbRecordSet as usual and then invoke the Delete method.

Method: TdbRecordSet:Delete

Type: void
Throws: TdbException

Java

```
void delete()
```

.Net

```
void Delete()
```

Deletes all records covered by the TdbRecordSet from the record set's target database(s). The record set can be prepared either with a physical database name or with a query, thus allowing records from multiple databases to be deleted.

Calling ASE Routines While Inserting or Updating a Record

An ASE (Application Software Exit) routine is a server-side function residing outside the TRIP kernel in an external library. Such routines are often custom-written and can be used at several predefined points, such as during loading of data into a database from a TFORM file.

From version 2.0-3 of TRIPnxp and TRIPjxp it is also possible to specify a list of ASE routines to be called upon commit of a new or modified record. In order for this to work, the TRIPsystem version used must be 6.2-4 or later.

ASE List Properties on the TdbRecord Class

There are two properties on the TdbRecord class are lists of ASE call information. There is one such list property for ASE routines to be called when a new record is committed, and one list property for ASE routines to be called when changes to an existing record are committed.

Property: TdbRecord:InsertAseList

Type: List<TdbAseCall>
Access: Read

Java

```
List<TdbAseCall> insertAseList()
```

.NET

```
List<TdbAseCall> InsertAseList { get; }
```

Property: TdbRecord:UpdateAseList

Type: List<TdbAseCall>
Access: Read

Java

```
List<TdbAseCall> updateAseList()
```

.NET

```
List<TdbAseCall> UpdateAseList { get; }
```

While these two lists are read-only properties, the contents of the lists themselves can be freely modified by the application.

Note that when calling the Clear() method on the TdbRecord instance, the contents of these lists is retained.

The TdbCallAse Class

The ASE call information is represented by instances of the TdbAseCall class. Mandatory information is the name of the ASE routine to call. Since an ASE can take an optional, single string argument, it is also possible to provide that.

Class: TdbAseCall

Derived from: Object
Located in: data

Constructor: TdbAseCall*Java*

```
TdbAseCall (String name)
```

```
TdbAseCall (String name, String argument)
```

.Net

```
TdbAseCall (String name)
```

```
TdbAseCall (String name, String argument)
```

The first constructor creates a TdbAseCall instance for an ASE that does not take any arguments. The second constructor creates a TdbAseCall instance for an ASE that takes an argument.

Use the TdbAseCall class together with the TdbRecord class like this:

Java

```
// Data population steps omitted for clarity
record.insertAseList(new TdbAseCall("myinsertase"));
record.commit();
```

VB.Net

```
// Data population steps omitted for clarity
record.InsertAseList.Add(new TdbAseCall("myinsertase"))
record.Commit()
```

Writing an ASE Routine

An ASE routine is written in the C programming language and linked as a dynamic library (DLL or shared object, depending on operating system).

Please refer to Appendix B in the TRIPmanager Administration Guide for detailed information about how to write ASE routines.

Error Messages from ASE Routines

If the ASE succeeds, it must return ASE_SUCCESS. If it fails, it absolutely must return the code ASE_FAIL. The application will then receive the error message for the last TRIPapi function called. However, if the ASE wishes to specify its own error message, it can use the TdbMessage function like this:

C

```
void set_error_message(char* msg, int msglen)
{
    TdbMessage(MSG_SET_ERROR,msg,&msglen);
}
```

9. Tuple Lists

Tuple lists enable the programmer to work with a set of fields in a record through a table-like abstraction layer.

Class: TdbTupleList

Derived from: Object
Located in: data

Class: TdbTuple

Derived from: Object
Located in: data

Instead of having to use separate TdbStructuredField instances to manage associated lists of sub fields, the TdbTupleList can be used instead. An added benefit is that when the TdbTupleList is used to manage sub fields, all fields in the tuple list are guaranteed to have the same number of sub fields.

Creating a tuple list

Tuple lists are always specific to a TdbComponent, i.e. the head or a specific part of a record. All fields except STRING and TEXT fields can be managed by a TdbTupleList.

Specifying a tuple list using a field group

From TRIP version 8.0 it is possible to create tuple lists based on field group definitions in the database design.

Java

```
TdbComponent h = record.getHead();
TdbTupleList lst = new TdbTupleList(session, h, db, "ADDR");
```

VB.Net

```
Dim h as TdbComponent = record.Head
Dim lst As New TdbTupleList(session ,h, db, "ADDR")
```

Specifying a tuple list explicitly

Tuple lists can also be specified explicitly without a field group in the db design:

Java

```
TdbComponent h = record.getHead();
TdbTupleList lst = new TdbTupleList(session,h,"NAME;ADDRESS");
```

VB.Net

```
Dim h as TdbComponent = record.Head
Dim lst As New TdbTupleList(session,h,"NAME;ADDRESS")
```

Ensuring presence of fields

Note that the fields to be included in the tuple list are not be present in the TdbComponent, they must first be added. This is the case when a new record is to be created:

Java

```
TdbRecord rec = new TdbRecord(session, "my_db", false);
TdbComponent h = rec.getHead();

h.createField("name",TdbFieldType.PhraseField);
h.createField("address",TdbFieldType.PhraseField);

TdbTupleList lst = new TdbTupleList(session,h,"NAME;ADDRESS");
```

VB.Net

```
Dim rec As new TdbRecord(session,"my_db",False)
Dim h as TdbComponent = rec.Head

h.CreateField("name",TdbFieldType.PhraseField)
h.CreateField("address",TdbFieldType.PhraseField)

Dim lst As New TdbTupleList(session,h,"NAME;ADDRESS")
```

New tuples

There are two methods available that support adding new tuples to a tuple list; the Append method and the Insert method.

Method: TdbTupleList:Append

Type: TdbTuple
Throws: TdbException

Java

```
TdbTuple append()
```

.Net

```
TdbTuple Append()
```

The Append method takes no parameters and returns a new TdbTuple object. The returned object is cached within the tuple list, and all subsequent requests for the newly appended tuple will be routed to the same object.

Method: TdbTupleList:Insert

Type: TdbTuple
Throws: TdbException

Java

```
TdbTuple insert(int index)
```

.Net

```
TdbTuple Insert(int index)
```


The insert method inserts a new, blank tuple at the specified zero-based index location. The returned object is cached within the tuple list, and all subsequent requests for the newly appended tuple will be routed to the same object.

The following example illustrates the Append and Insert methods.

Java

```
TdbComponent h = record.getHead();
TdbTupleList lst = new TdbTupleList(session,h,"NAME;ADDRESS");

TdbTuple t1 = lst.append(); // Add a new tuple
TdbTuple t2 = lst.insert(0); // New tuple first in list

t1.setValue(0,"John Doe");
t1.setValue(1,"Elm Street");
t2.setValue("NAME","Jane Doe");

record.commit(); // save changes
```

VB.Net

```
Dim rec As new TdbRecord(session,"my_db",False)
Dim h As TdbComponent = rec.Head
Dim lst As New TdbTupleList(session,h,"NAME;ADDRESS")

Dim t1 As TdbTuple = lst.Append() ' Add new tuple
Dim t2 As TdbTuple = lst.Insert(0) ' New tuple first in list

t1.SetValue(0,"John Doe")
t1.SetValue(1,"Elm Street")
t2.SetValue("NAME","Jane Doe")

record.Commit()
```

Accessing tuples

The TdbTupleList properties RowCount and ColumnCount together with the get/indexer method can be used to iterate through a tuple list.

Java

```
int rowidx,colidx;
TdbTuple tuple;
String tval;

for (rowidx=0;rowidx<tupleList.getRowCount();rowidx++)
{
    tuple = tupleList.get(rowidx);
    for (colidx=0;colidx<tupleList.getColumnCount();colidx++)
    {
        tval = tuple.getValue(colidx);
    }
}
```

VB.NET

```
Dim rowidx As Integer
Dim colidx As Integer
Dim tuple As TdbTuple
Dim tval As String

For rowidx = 0 To tupleList.RowCount
    tuple = tupleList.Get(rowidx)
    For colidx = 0 To tupleList.ColumnCount
        tval = t.GetValue(j)
    Next colidx
Next rowidx
```

Clearing tuples

To clear a tuple means to set all subfield values for the tuple to empty. This will not remove the tuple from the tuple list. The Clear method is available in the TdbTuple class.

Method: TdbTuple:Clear

Type: void
Throws: TdbException

Java

```
void clear()
```

.Net

```
void Clear()
```

Removing tuples

Tuples can be removed from the tuple list using the Remove method on TdbTupleList or on TdbTuple.

Method: TdbTuple:Remove

Type: void
Throws: TdbException

Java

```
void remove()
```

.Net

```
void Remove()
```

The Remove method on the TdbTupleList takes the zero-based tuple index as an argument.

Method: TdbTupleList:Remove

Type: void
Throws: TdbException

Java

```
void remove(int index)
```

.Net

```
void Remove(int index)
```

Removed tuples to which the application still holds references will throw a TdbException upon access. The application should therefore immediately discard any TdbTuple object that has been removed.

10. Managing databases and thesauri

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

The management of databases revolves around a core class and a derivation

Class: TdbDatabaseDesign

Derived from: TdbSerializableObject
Located in: database

Class: TdbThesaurusDesign

Derived from: TdbDatabaseDesign
Located in: database

The base class, TdbDatabaseDesign, provides all of the functionality whilst the derived class, TdbThesaurusDesign, provides type-specific extensions to ensure that when creating or retrieving designs, TRIP is initialized to correctly interpret the design as a thesaurus.

As with all library classes, the majority of interaction with the design of a database or thesaurus takes place on locally cached data; only once the application chooses to commit any changes made locally is a network transaction performed to store the resulting design to the server.

Creating a new database or thesaurus

In order to create a new design, simply create an object of the appropriate class, fill its properties as required and then commit the design to the server. For example, the following example shows how to create a new database:

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);

// Set default DB design properties
db.get("sample",true);

db.setBafFile("DATABASES:SAMPLE.BAF");
db.setBifFile("DATABASES:SAMPLE.BIF");
db.setVifFile("DATABASES:SAMPLE.VIF");

db.put("sample");
```

VB.Net

```
Dim db As New TdbDatabaseDesign(session)

// Set default DB design properties
db.Get("sample", True)

db.BafFile = "DATABASES:SAMPLE.BAF"
db.BifFile = "DATABASES:SAMPLE.BIF"
db.VifFile = "DATABASES:SAMPLE.VIF"

db.Put("sample");
```

This simple example creates a skeleton of a database. To make a more useful database, the application should add a collection of field designs to the database.

Class: TdbFieldDesign

Derived from: Object
Located in: database

Instances of field designs are added to a database design using the following method

Method: TdbDatabaseDesign:AddField

Type: void
Throws: N/A

Java

```
void addField(TdbFieldDesign field)
```

.Net

```
void AddField(TdbFieldDesign field)
```

Add a copy of the provided field design to the database field list. The field design object provided can be reused without impacting the underlying database design due to the copying of the field that occurs during addition.

Applications managing the entire field collection at once can use a collection property that explicitly clears any pre-existing notion of a field collection and replaces it with a copy of the application's version.

Collection: TdbDatabaseDesign:Fields

Type: List of TdbFieldDesign
Access: Read, Write

Java

```
List<TdbFieldDesign> fields()
void putFields(Collection<TdbFieldDesign> fields)
```

.Net

```
ICollection Fields { get; set; }
```

Retrieve and replace the collection of field designs associated with the database design.

The following example expands on the example above to add two fields to the database design before attempting to create it:

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);

db.setBafFile("DATABASES:SAMPLE.BAF");
db.setBifFile("DATABASES:SAMPLE.BIF");
db.setVifFile("DATABASES:SAMPLE.VIF");

TdbFieldDesign field = new TdbFieldDesign();

field.setName("Field_1");
field.setType("phrase");
db.addField(field);

// Note that we reuse the same field design object, as the
// object gets copied into the underlying database design
field.setName("Field_2");
field.setType("text");
db.addField(field);

db.put("SAMPLE");
```

VB.Net

```
Dim db as New TdbDatabaseDesign(session)
```

```
db.BafFile = "DATABASES:SAMPLE.BAF"
```

```
db.BifFile = "DATABASES:SAMPLE.BIF"
```

```
db.VifFile = "DATABASES:SAMPLE.VIF"
```

```
Dim field As New TdbFieldDesign
```

```
field.Name = "Field_1"
```

```
field.Type = "phrase"
```

```
db.AddField(field)
```

' Note that we reuse the same field design object, as the
' object gets copied into the underlying database design

```
field.Name = "Field_2"
```

```
field.Type = "text"
```

```
db.AddField(field)
```

```
db.Put("sample")
```

Modifying an existing database or thesaurus

In order to modify an existing design, simply retrieve that design from the server, modify as required and then commit the design again. For example:

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);

db.get("SAMPLE");
db.setDescription("Simple update to database");
db.put();
```

VB.Net

```
Dim db As New TdbDatabaseDesign(session)

db.Get("sample")
db.Description = "Simple update to database"
db.Put()
```

To modify the field collection, first retrieve all fields from the design then make whatever modifications are required before storing the complete field collection back in the design. Finally, commit the database design back to the server, as shown below.

The field collection retrieved using the Fields property is a snapshot of the design at that point and cannot be used to automatically propagate changes onto the underlying design. Any changes made must be stored to the design explicitly.

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);

db.get("sample");
Collection<TdbFieldDesign> fields = db.fields();

// change the "fields" collection however is required
// e.g. modify existing field properties, add new fields, etc.

db.putFields(fields);
db.put();
```


VB.Net

```
Dim db as New TdbDatabaseDesign(session)
db.Get("sample")

Dim fields As IList = db.Fields

// change the "fields" collection how is required
// e.g. modify existing field properties, add new fields, etc.

db.Fields = fields
db.Put()
```

Note that there are many different get and put operations supported on the database design class that are not described explicitly here. All aspects of the database or thesaurus design can be interrogated and/or modified by the calling application. For more detail, consult the reference documentation provided.

Deleting fields

One exception to the rule of making all changes locally before committing the design is removing fields. To remove a field from an existing design, retrieve the design from the server, locate the field required and delete it. Applications should not mix updates and delete operations, as this will cause error conditions to become more likely.

Java

```
TdbDatabaseDesign db = new TdbDatabaseDesign(session);
TdbFieldDesign field;

db.get("SAMPLE");
field = db.getField("Field_1");
db.removeField(field);

// No need to commit the database design at this point, as the
// field has been removed on the server
```

VB.Net

```
Dim db as New TdbDatabaseDesign(session)
db.Get("sample")

' Use the .Net indexer property to get the field design that
' is to be deleted, and remove it from the design
db.RemoveField(db("Field_1"))

' No need to commit the database design at this point, as the
' field has been removed on the server
```

At this point, the field has been removed from the database design on the server and the TdbFieldDesign object on the client is detached from the database design, allowing it to be re-inserted if required, perhaps into a different database design.

Copying an existing database or thesaurus

There are two different kinds of copy operation supported for databases and thesauri. The methods involved are Copy and DeepCopy, the difference being the level of copy that's performed. During a normal copy operation, the database design is copied to a new name, but file definitions are reset to a default value, formats are not copied and database access rights are not established.

During a deep copy operation, in contrast, all formats are copied, all database access rights are copied, and the file definitions for the new copy are established using a consistent naming scheme to the old database, i.e. located in the same physical location but with a new name reflecting the name of the new database.

Method: TdbDatabaseDesign:Copy

Type: void
Throws: TdbException

Java

```
void copy(String newname, TdbControlObject newobj)
```

.Net

```
void Copy(String newname, TdbControlObject newobj)
```

Perform a shallow copy from the current database design to a new name. The current object must have been created using a valid Control object reference, or the Get method must have been invoked prior to attempting to copy the database to a new name.

If the “newobj” parameter is not null on input, and references a valid, blank, Control object reference, that reference will be updated by the Copy method to contain information relevant to the new database copy upon completion.

Method: TdbDatabaseDesign:DeepCopy

Type: void
Throws: TdbException

Java

```
void deepCopy(String newname, TdbControlObject newobj)
```

.Net

```
void DeepCopy(String newname, TdbControlObject newobj)
```

Perform a deep copy from the current database design to a new name. The deep copy retains database access rights, formats and file specification consistency with the original. The current object must have been created using a valid Control object reference, or the Get method must have been invoked prior to attempting to copy the database to a new name.

If the “newobj” parameter is not null on input, and references a valid, blank, Control object reference, that reference will be updated by the Copy method to contain information relevant to the new database copy upon completion.

Deleting existing databases and thesauri

To delete a database or thesaurus, simply create an object of the appropriate type and invoke the Delete method.

Method: TdbDatabaseDesign:Delete

Type: void
Throws: TdbException

Java

```
void delete(String name)  
void delete()
```

.Net

```
void Delete(String name)  
void Delete()
```

Delete the named database or thesaurus from the server. The second version of the method is only usable if the object was constructed using a valid Control object reference.



11. Managing formats

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

To retrieve a list of formats associated with a given database, use the appropriate list class, as shown in the following example.

Java

```
TdbDatabaseDesign db;

TdbOutputFormatList fmts;

db = new TdbDatabaseDesign(session);
db.get("alice");
fmts = new TdbOutputFormatList(session, db);
```

VB.Net

```
Dim db As New TdbDatabaseDesign(session)
db.Get("sample")

Dim fmts As New TdbOutputFormatList(session, db)
```

This uses the database design's associated Control object reference to seed the output format list. The list retrieved can be iterated upon in the normal manner, as explained in section 6.

Creating new formats

To create a new format, create a format object of the appropriate type, update its properties and then use the Create method to commit the new format to the server.

Method: TdbFormat.Create

Type: void
Throws: TdbException

Java

```
void create(String name, String database)
void create(String name)
```

.Net

```
void Create(String name, String database)
void Create(String name)
```

The first version of this method creates a format that is associated with a database, i.e. an entry form or an output format. The second version is used to create a format that is not associated with a database, i.e. a search form.

The following example shows how to create a new output format.

Java

```
TdbOutputFormat format = new TdbOutputFormat(session);
Vector<String> design = new vector<String>();

design.add("<");
design.add("  <box 1 at b(*)+1,1");
design.add("      <t=Hello world>");
design.add("  >");
design.add(">");

format.setContent(design);
format.setComment("A new output format");

format.create("new_form", "alice");
```

VB.Net

```
Dim format As New TdbOutputFormat(session)
Dim design As New ArrayList

design.Add("<")
design.Add("  <box 1 at b(*)+1,1")
design.Add("      <t=Hello world>")
design.Add("  >")
design.Add(">")

format.Content = design
format.Comment = "A new output format"

format.Create("new_form", "alice");
```

Modifying existing formats

Format designs are manipulated using the following classes.

Class: TdbEntryForm

Derived from: TdbFormat
Located in: forms

This class encapsulates the design and properties of a TRIPclassic entry form. The class provides no special parsing or interpretation functions for dealing with the form definition.

Class: TdbOutputFormat

Derived from: TdbFormat
Located in: forms

This class encapsulates the design and properties of an output format. In addition to the standard manipulation methods provided by the base TdbFormat class, output formats can be exercised in a test mode using a method on this class. This test mode creates a temporary format on the server, runs some output using the temporary format and then deletes the format before sending the resulting data back to the client. This can be extremely useful when testing modifications to production reports without impacting the user community in any way.

Class: TdbSearchForm

Derived from: TdbFormat
Located in: forms

This class encapsulates the design and properties of a TRIPclassic search form. The class provides no special parsing or interpretation functions for dealing with the form definition.

Once the desired format is identified, its properties and design information can be retrieved from the server, as shown in the following example.

Java

```
TdbOutputFormat format = new TdbOutputFormat(session);  
format.get("full", "alice");
```

VB.Net

```
Dim format As New TdbOutputFormat(session)  
format.Get("full", "alice")
```

The definition of the format, i.e. its content, can be retrieved and/or modified using the following collection property of the base TdbFormat class. This property does not provide "live" access to the underlying design, but rather provides a snapshot that can be edited and then committed to update the design.

Property: TdbFormat:Content

Type: List of String
Access: Read, Write

Java

```
Collection<String> content()  
void setContent(Collection<String>)
```

.Net

```
ICollection Content { get; set; }
```

Note that the collection returned by the .Net version can be freely cast to an `ArrayList` for modification.

The following example shows this collection property being used.

Java

```
TdbOutputFormat format;  
Collection<String> design;  
  
// Retrieve the format from the server  
format = new TdbOutputFormat(session);  
format.get("full", "alice");  
  
// Extract the format's design  
design = format.content();  
  
// modify the format's design  
editDesign(design);  
  
// update the format's design and properties  
format.setContent(design);  
format.setComment("updated to reflect new requirements.");  
  
// commit the format to the server  
format.put();
```


VB.Net

```

Dim format As New TdbOutputFormat(session)

' Retrieve the format from the server
format.Get("full", "alice")

' Extract the format's design
Dim design As IList = format.Content

' Fill a text box with the design, or run a wizard,
' or whatever
...

' Store the modified design, commit the format to the server
format.Content = design
format.Put()

```

Testing output formats

As previously mentioned, output formats can be tested for validity prior to a final commit. This involves the server in creating a temporary format and attempting to use that format. Using this test operation allows a form designer to make modifications to a production format without affecting their end users until all modifications are complete.

To test an output format, use the Test method:

Method: TdbOutputFormat:Test

Type: void
Throws: TdbException

Java

```
void test(TdbKernelWindow window)
```

VB.Net

```
void Test(TdbKernelWindow window)
```

Test the current format, sending any output produced by the format under test to the provided kernel window structure (see Section 4 for more detail on kernel window buffers).

The following example shows how to use the Test method to validate changes to a format before committing.

Java

```
TdbOutputFormat format = new TdbOutputFormat(session);
TdbKernelWindow window = new TdbKernelWindow(session);

// Retrieve the format from the server
format.get("production_fmt", "database");

// change the format, getting and putting the
// Content collection
editFormat(format);

// test changes to the format's content
format.test(window);

// show the results
System.out.println(window.toString());

// store the final result
if( confirmWithUser() )
    format.put();
```

VB.Net

```

Dim format As New TdbOutputFormat(session)
Dim window As New TdbKernelWindow(session)

' Retrieve the format from the server
format.Get("production_fmt", "database")

' Edit the format
...

' Test the changes made, show the results
format.Test(window)
myQuickViewer.Text = window.ToString()

' Commit the result
format.Put()

```

Deleting existing formats

To delete a format, create a format object of the appropriate type and then use the Delete method to commit the deletion to the server.

Method: TdbFormat.Delete

Type: void
Throws: TdbException

Java

```

void delete(String format, String database)
void delete(String format)
void delete()

```

VB.Net

```

void Delete(String format, String database)
void Delete(String format)
void Delete()

```

The first version of the method supports deletion of formats that are associated with a database, i.e. entry or output formats. The second version supports deletion of formats that are not associated with a database, i.e. search forms. The third version supports deletion of any kind of format provided the object in question was constructed using a valid Control object reference for the format.

12. Managing database clusters

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Database clusters, which are predefined collections of databases and/or thesauri that are always opened together, are managed using the TdbDatabaseClusterDesign class.

Class: TdbDatabaseClusterDesign

Derived from: TdbSerializableObject
Located in: database

Creating a new cluster

To create a new cluster, simply create a TdbDatabaseClusterDesign object, set its properties and membership list, then commit the new cluster to the server using the Put method.

Collection: TdbDatabaseClusterDesign:Members

Type: TdbControlObjectList
Access: Read, Write

Java

```
TdbControlObjectList members()
void putMembers(TdbControlObjectList members)
```

.Net

```
TdbControlObjectList Members { get; set; }
```

The membership list retrieved from this collection is a snapshot of the actual membership list that must be stored back to the object in order to make any difference to the cluster design.

Method: TdbDatabaseClusterDesign:Put

Type: void
Throws: TdbException

Java

```
void put(String name)
void put()
```

.Net

```
void Put(String name)
void Put()
```

Store a new or modified cluster definition to the server. The second version of the method is only usable if the object was constructed using a valid Control object reference for the database cluster in question, or if the design was been retrieved from the server using the Get method documented below.

To add a member to the cluster, use the Add method.

Method: TdbDatabaseClusterDesign:Add

Type: void
Throws: N/A

Java

```
void add(String cluster)
void add(TdbControlObject cluster)
```

.Net

```
void Add(String cluster)
void Add(TdbControlObject cluster)
```

The first version of the method is most useful for adding by name, whilst the second version is intended for adding when picking from a list generated by a TdbControlObjectList request.

The following example shows how to create a new database cluster.

Java

```
TdbDatabaseClusterDesign cls = new
TdbDatabaseClusterDesign(session)
```

```
// Create the membership list
cls.add("alice");
cls.add("carroll");
```

```
// Create the new cluster
cls.put("my_cluster");
```

VB.Net

```
Dim cls As New TdbDatabaseClusterDesign(session)
```

```
' Create the membership list
cls.Add("alice")
cls.Add("carroll")
```

```
' Create the new cluster
cls.Put("my_cluster")
```

Modifying existing clusters

To modify the definition of a cluster, simply create a TdbDatabaseClusterDesign object, use the Get method to retrieve the existing definition, modify as required and store the new definition using the Put method.

Method: TdbDatabaseClusterDesign:Get

Type: void
Throws: TdbException

Java

```
void get(String clusterName)  
void get()
```

.Net

```
void Get(String clusterName)  
void Get()
```

The second version of this method requires that the object was constructed with a valid Control object reference for the cluster being addressed.

Deleting existing clusters

To delete an existing database cluster, simply create a TdbDatabaseClusterDesign object and invoke the Delete method.

Method: TdbDatabaseClusterDesign:Delete

Type: void
Throws: TdbException

Java

```
void delete(String name)  
void delete()
```

.Net

```
void Delete(String name)  
void Delete()
```

Delete an existing cluster definition from the server. The second version of the method is only usable if the object was constructed using a valid Control object reference, or if the Get method was used to retrieve an existing design prior to deletion.

13. Managing classification schemes

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Classification schemes are special purpose databases, managed almost entirely by the system and whose content is entirely dictated by the subsystem used to create the scheme. Classification subsystems are kernel modules that enable the use of a specific classification algorithm.

Class: TdbClassificationScheme

Derived from: TdbMessageProvider
Located in: classification

Creating a new scheme

In order to create a new scheme, the application must know which subsystem it wishes to use for that scheme. Different subsystems can offer widely different classification capabilities, classification types and end user interaction capabilities, this it is very important to choose an appropriate subsystem, or let the DBA choose such during the creation process.

The collection of available subsystems is always available on the TdbSession object.

Collection: TdbSession.SubsystemNames

Type: List of String
Access: Read

Java

```
List<String> subsystemNames()
```

.Net

```
ICollection SubsystemNames { get; }
```

Retrieve a collection of classification subsystem names.

Having chosen the name of an appropriate subsystem, the calling application can dereference that name to a unique ID, which is required for all other interaction, again using the TdbSession object.

Method: TdbSession.GetSubsystemID

Type: Integer
Throws: N/A

Java

```
int getSubsystemID(String subsystemName)
```

.Net

```
Int32 GetSubsystemID(String subsystemName)
```

Retrieve the kernel-specific unique ID for the named subsystem. Note that there is no guarantee between installations, or even between sessions, that the ID retrieved for a given name will be identical; applications should always use this method to retrieve the unique ID for the named subsystem.

Having chosen a subsystem with which to create the new scheme, simply create an object of type `TdbClassificationScheme` and invoke the `Create` method. The new classification scheme's data files will be created in the directory referenced by the system logical name `TDBS_CLS`, although this can be modified later, if required.

Method: `TdbClassificationScheme.Create`

Type: `void`
Throws: `TdbException`

Java

```
void create(String name, int subsystem, int max_items)
```

.Net

```
void Create(String name, Int32 subsystem, Int32 max_items)
```

Create a new scheme, using the defined subsystem and name. The "max_items" parameter refers to the maximum number of training items per category that will be processed by the subsystem.

The following example shows how to create a new scheme.

Java

```
TdbClassificationScheme cls = new TdbClassificationScheme(session)

// Pick the first available subsystem
String name = session.subsystemNames().get(0);
int id = session.getSubsystemId(name);

// Create a new scheme
cls.create("my_new_scheme", id, 100);
```

VB.Net

```
Dim cls As New TdbClassificationScheme(session)

' Pick the first available subsystem
Dim name as String = session.SubsystemNames(0)
Dim id As Integer = session.GetSubsystemId(name)

' Create a new scheme
cls.Create("my_new_scheme", id, 100)
```

Modifying an existing scheme

To modify the attributes of an existing scheme, for example the location used to store its files, simply create an object of type `TdbClassificationScheme`, retrieve the current design using the `Get` method, modify whatever properties are required and then commit the changes using the `Put` method.

Method: TdbClassificationScheme:Get

Type: void
Throws: TdbException

Java

```
void get(String name)  
void get()
```

.Net

```
void Get(String name)  
void Get()
```

Retrieve a scheme's properties from the server. In order to use the second version of the method, the object must have been constructed with a valid Control object reference for the scheme in question.

Note that when modifying file locations, the DBA is responsible for moving any existing files relating to the scheme to the new location. Failure to do this will result in the scheme potentially being unusable, as schemes may require certain data to be present even in an "empty" scheme.

Method: TdbClassificationScheme:Put

Type: void
Throws: TdbException

Java

```
void put()
```

.Net

```
void Put()
```

Store any modifications to the scheme's design to the server. This method can only be called after a successful invocation of the Get method.

The following example shows how to modify an existing scheme.

Java

```
TdbClassificationScheme cls = new TdbClassificationScheme(session);

cls.get("my_scheme");

// Change properties as required
cls.setDescription("New comment");
cls.setLocation("MYSCHEMES");

// Store the modified design to the server
cls.put();
```

VB.Net

```
Dim cls As New TdbClassificationScheme(session)

cls.Get("my_scheme")

' Change the properties as required
cls.Description = "New Comment"
cls.Location = "MYSCHEMES"

' Store the modified design to the server
cls.Put()
```

Note that as shown here, all files for a given scheme are located using a single logical name. There is no explicit capability to set the file names for each file comprising a scheme separately due to the fact that a scheme may consist of an arbitrary number of files over and above the normal BAF/BIF/VIF.

Managing categories within a scheme

In order to classify information, a scheme needs to be trained in how to recognize the different types, or categories, of information in which the user is interested. Each category of information has certain properties, such as its name and an optional comment, and a collection of data with which it has been trained. Adding new information to the training collection helps the scheme to recognize information that belongs to the category, and to distinguish that category from others in the same scheme.

In order to interact with categories, the developer uses the TdbCategory class.

Class: TdbCategory

Derived from: TdbMessageProvider
Located in: classification

Creating new categories

To create a new category, use the Add factory method on the appropriate scheme object. This creates a new category, adds it to the scheme, links the local representation of the category into the scheme, and returns the new category to the caller.

Method: TdbClassificationScheme:Add

Type: TdbCategory
Throws: TdbException

Java

```
TdbCategory add(String name, String comment)
```

.Net

```
TdbCategory Add(String name, String comment)
```

Create a new category within the scheme with the properties as defined. The comment parameter is optional and can be set to null if there is no comment required.

In contrast to the capabilities the server makes available, category names must be unique in order to be manipulated with the TRIPjxp and TRIPnpx libraries. Underlying each category is the actual unique ID assigned by the server, which can be retrieved using the Id property.

Property: TdbCategory:Id

Type: Integer
Access: Read

Java

```
int getId()
```

.Net

```
Int32 Id { get; }
```

Retrieve the unique ID assigned by the server to this category. This value has no specific meaning, although schemes may equate the category ID to a record number in the scheme database.

Retrieving existing categories

Following the successful invocation of the Get method on the scheme, the collection of categories that have already been defined is retrieved from the scheme using the fact that the scheme itself is a collection of categories, and can be enumerated appropriately.

Java

```
TdbClassificationScheme cls = new TdbClassificationScheme(session);

// Retrieve the scheme from the server
cls.get("my_scheme");

// Iterate over the categories embodied by the scheme
for( TdbCategory category : cls )
{
    ...
}
```

VB.Net

```
Dim cls As New TdbClassificationScheme(session)

' Retrieve the scheme from the server
cls.Get("my_scheme")

' Iterate over the categories embodied by the scheme
For Each category As TdbCategory In cls
    ...
Next
```

Individual categories may be retrieved from a scheme using the `GetCategory` method, provided that the scheme's information has already been retrieved from the server using the `Get` method.

Method: TdbClassificationScheme:GetCategory

Type: TdbCategory
Throws: TdbException

Java

```
TdbCategory getCategory(String name)
TdbCategory getCategory(int uniqueId)
```

.Net

```
TdbCategory GetCategory(String name)
TdbCategory GetCategory(int uniqueId)
```

Retrieve the category instance with either the name or the unique ID specified. Note that .Net also provides a class indexer mirroring each of these methods. If the named or identified category is not found, the methods return null.

Training a category

Once the category is created or retrieved, applications can train the category with text that typifies the information represented by that category.

To train a category, simply invoke the Train method on the appropriate TdbCategory instance.

Method: TdbCategory:Train

Type: void
Throws: TdbException

Java

```
void train(String data, String filename, TdbTrainingInfo info)
```

.Net

```
void Train(String data, String filename, TdbTrainingInfo info)
```

Add training data to the category, updating "info" if not null with the result.

When providing training data, the calling application can provide a filename from which the data was read, simply for labeling purposes. If no filename (or URL, or any kind of location reference) is involved, the calling application should provide some hopefully meaningful label that will make sense to the user in charge of the scheme.

If the TdbTrainingInfo instance is non-null on input, the method will update the instance with information created by the training process.

Java

```
TdbClassificationScheme cls = new TdbClassificationScheme(session);

cls.get("my_scheme");

// Create a new category
TdbCategory newcat = cls.add("new category", null);

// Acquire the data with which we're going to train
String file = "/usr/local/training/data1.txt";
String data = getDataFromFile(file);

// Create an info instance
TdbTrainingInfo info = new TdbTrainingInfo();

// Train the category
newcat.train(data, file, info);

// Output training information
System.out.println("Category: " + newcat.getName() +
    " trained with " + info.getName() +
    " generated: " + info.getComment());
```

VB.Net

```
Dim cls As New TdbClassificationScheme(session)

cls.Get("my_scheme")

' Create a new category
Dim newcat As TdbCategory = cls.Add("new category", Nothing)

' Acquire the data with which we're going to train
Dim file As String = "C:\training\data1.txt"
Dim data As String = getDataFromFile(file)

' Create an info instance
Dim info As New TdbTrainingInfo()

' Train the category
newcat.Train(data, file, info)

' Output training information
Dim msg As String = "Category: " & newcat.Name & _
    " trained with " & info.Name & _
    " generated: " & info.Comment

MsgBox(msg)
```

Viewing training material for a category

The collection of labels and comments for training information already submitted for a category is available by enumerating the category itself, which yields a collection of `TdbTrainingInfo` instances, one for each item of training data submitted to that category.

Class: TdbTrainingInfo

Derived from: Object
Located in: classification

Objects of this class contain the name and optional description of an item of training data.

The following example shows how to enumerate a category to view the training information.

Java

```
TdbClassificationScheme cls = new TdbClassificationScheme(session);

// Retrieve the classification scheme from the server
cls.get("my_scheme");

// Enumerate the categories in the scheme
for( TdbCategory category : cls )
{
    // Enumerate the training info in each category
    for( TdbTrainingInfo info : category )
    {
        ...
    }
}
```

VB.Net

```
Dim cls As New TdbClassificationScheme(cls)

' Retrieve the classification scheme from the server
cls.Get("my_scheme")

' Enumerate the categories in the scheme
For Each category As TdbCategory In cls

    ' Enumerate the training info in each category
    For Each info As TdbTrainingInfo In category
        ...
    Next

Next
```

There are only two properties currently recorded as training information: the label, or filename, provided when submitting the training data, and a comment or description provided by the classification algorithm.

Removing training for a category

To completely remove all training material for a category, without deleting the category itself, invoke the `Untrain` method on the `TdbCategory` object.

Method: Untrain

Type: void
Throws: TdbException

Java

```
void untrain()
```

.Net

```
void Untrain()
```

Deleting an existing category

In order to delete a category, invoke the Remove method on the scheme to which the category is attached.

Method: TdbClassificationScheme:Remove

Type: void
Throws: TdbException

Java

```
TdbCategory remove(String name)  
void remove(TdbCategory category)
```

.Net

```
TdbCategory Remove(String name)  
void Remove(TdbCategory category)
```

This method removes the category from its associated scheme on both client and server. The first version returns the removed category simply for reference.

Testing classification

Once the categories within the scheme have been trained, it is very useful to be able to test the classification algorithm before attempting to classify databases. To test a scheme with some text data, simply invoke the Classify method on the TdbClassificationScheme object.

Method: TdbClassificationScheme:Classify

Type: void
Throws: TdbException

Java

```
void classify(String data, List<TdbCategory> cats)
```

.Net

```
void Classify(String data, IList cats)
```

Test the classification scheme with the provided data. On completion, the list of categories will be populated with the categories to which the data was assigned by the algorithm.

Using an existing TRIP database to create and train categories

As an alternative to creating training data in text files or any other environment outside of TRIP, the classification scheme can instead be trained directly from an existing TRIP database. For this to work, the database in question must have a field in its design that holds the name of the category to which the text in the record is assigned.

Method: `TdbClassificationScheme.Infer`

Type: `void`
Throws: `TdbException`

Java

```
void infer(String database, String field, int max)
```

.Net

```
void Infer(String database, String field, int max)
```

Using this method, all the records in the named database are read, the named field's contents are extracted and treated as the category name to create / train, up to a maximum number of 'max' records per category. All of the text found in fields flagged for inclusion in non-Boolean indexing is used as training material for that category, with the label being generated as the database name and record number.

Deleting an existing classification scheme

In order to delete an existing classification scheme, simply invoke the `Delete` method on the `TdbClassificationScheme` object. Note that in contrast to normal TRIP databases, the scheme does not have to be empty in order for this operation to succeed, as the classification subsystem is responsible for removing all data from the scheme before the scheme is actually deleted.

Method: `TdbClassificationScheme.Delete`

Type: `void`
Throws: `TdbException`

Java

```
void delete()
```

.Net

```
void Delete()
```

Delete an existing classification scheme. Note that the object must have either been created using a valid `Control` object reference for the scheme in question, or the scheme must have been retrieved from the server using the `Get` method in order for this method to succeed.

13. Managing access rights

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

All management of access rights, whether to a database, thesaurus or cluster, and whether for a user or group, is accomplished using the TdbDatabaseAccess class.

Class: TdbDatabaseAccess

Derived from: TdbMessageProvider
Located in: database

The only operations valid to perform using this class are to retrieve a user or group's access to a database, thesaurus or cluster, or to update their access. The former operation uses the Get method, whilst the latter operation uses the Put method.

Method: TdbDatabaseAccess:Get

Type: void
Throws: TdbException

Java

```
void get(String user_or_group, String database)
void get(String user_or_group)
```

.Net

```
void Get(String user_or_group, String database)
void Get(String user_or_group)
```

Retrieve the access rights for the named user or group to the named database, thesaurus or cluster. The second form of the method is only usable if the object was constructed using a valid Control object reference for the database in question.

Method: TdbDatabaseAccess:Put

Type: void
Throws: TdbException

Java

```
void put(String user_or_group, String database)
void put()
```

.Net

```
void Put(String user_or_group, String database)
void Put()
```

Commit new access rights for the named user or group to the named database, thesaurus or cluster. The second form of the method is only usable if the Get method has preceded a call to this method, establishing the context of the operation.

Working with access rights

Assuming the application has called the Get method to retrieve the current access rights of a user to a database, the application can now interrogate and modify those access rights as required. Obviously this only works if the calling user is the FM of the database in question.

Each of the types of access has an associated property that supports retrieval and modification of the underlying database access privilege, as documented below. Note that whilst the documentation here regards properties for read access, the same properties exist for write access.

Property: TdbDatabaseAccess:ReadAccess

Type: TdbAccessRights
Access: Read, Write

Java

```
TdbAccessRights getReadAccess()  
void setReadAccess(TdbAccessRights rights)
```

.Net

```
TdbAccessRights ReadAccess { get; set; }
```

This property sets the overall read access mode for the user/group. The available types of access, as taken from the TdbAccessRights enumeration, are: None, Selected and All. If the mode chosen is Selected, then the ReadFields property must be used to determine the fields within the database to which the user has access.

Property: TdbDatabaseAccess:ReadFields

Type: List of Integer
Access: Read, Write

Java

```
List<Integer> readFields()  
void setReadFields(Collection<Integer> fields)
```

.Net

```
ICollection ReadFields { get; set; }
```

Retrieve and modify the set of fields from the database or thesaurus design to which the user or group has read access. The entries within the list returned or provided correspond to field numbers within the database or thesaurus design. Not valid for use with a cluster.

Property: TdbDatabaseAccess:ReadScope

Type: String
Access: Read, Write

Java

```
String getReadScope()  
void setReadScope(String scope)
```

.Net

```
String ReadScope { get; set; }
```

Retrieve and modify the content-specific restriction that is to be applied to the user's available record set for the database, thesaurus or cluster.

The following example shows how to work with access rights.

Java

```
// Retrieve the database design for which we're going  
// to be modifying access rights  
TdbDatabaseDesign db = new TdbDatabaseDesign(session)  
db.get("alice");  
  
// Establish an access rights container associated with the  
// database  
TdbDatabaseAccess acc = new TdbDatabaseAccess(session, db);  
acc.get("my_user");  
  
// Grant the user unrestricted read access  
acc.setReadAccess(TdbAccessRights.All);  
  
// Grant the user selected write access to Chapter and Speaker  
acc.setWriteAccess(TdbAccessRights.Selected);  
ArrayList<Integer> fields = new ArrayList<Integer>;  
fields.add(db.getFieldByName("chapter").getNumber());  
fields.add(db.getFieldByName("speaker").getNumber());  
acc.setWriteFields(fields);  
  
// Store the new rights back to the server  
acc.put();
```

VB.Net

```
' Retrieve the database design for which we're going
' to be modifying access rights
Dim db As New TdbDatabaseDesign(session)
db.Get("alice")

' Establish an access rights container associated with the database
Dim acc As New TdbDatabaseAccess(session, db)
acc.Get("my_user")

' Grant the user unrestricted read access
acc.ReadAccess = TdbAccessRights.All

' Grant the user selected write access to Chapter and Speaker
acc.WriteAccess = TdbAccessRights.Selected
Dim fields As New ArrayList()
fields.Add(db["chapter"].Number)
fields.Add(db["speaker"].Number)
acc.WriteFields = fields

' Store the new rights back to the server
acc.Put()
```

15. Managing users

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Users are managed using the following class.

Class: TdbUser

Derived from: TdbSerializableObject
Located in: users

Creating new users

To create new users, simply create a TdbUser object and then invoke the following method:

Method: TdbUser:Create

Type: void
Throws: TdbException

Java

```
void create(String username, String password)
```

.Net

```
void Create(String username, String password)
```

Create the named user with the defined password. This password is encrypted for transmission to the server. If the attempt succeeds, a new user will be created, owned by the calling user. The new user will not have any management privilege, will not have any specific database access rights, and will be a member of the PUBLIC user group.

Prior to calling this method, the application may set any number of properties related to the user's profile, for example their desired date format, their real name, etc. All such properties will be transmitted to the server along with the request to create the new user.

If the calling user is the SYSTEM user, create user and/or file managers by setting the appropriate privilege properties before creating the user.

Property: TdbUser:IsUM

Type: Boolean
Access: Read, Write

Java

```
boolean getIsUM()  
void setIsUM(boolean um)
```

.Net

```
Boolean IsUM { get; set; }
```

Define or retrieve the user's UM privilege level. Setting this value only has effect if the calling user is SYSTEM.

Property: TdbUser:IsFM

Type: Boolean
Access: Read, Write

Java

```
boolean getIsFM()  
void setIsFM(Boolean fm)
```

.Net

```
Boolean IsFM { get; set; }
```

Define or retrieve the user's FM privilege level. Setting this value only has effect if the calling user is SYSTEM.

The following example shows a new unprivileged user being created.

Java

```
TdbUser user = new TdbUser(session);  
  
user.setRealName("My New User");  
user.setLoginProcedure("Public.Startup");  
user.create("newuser", "newpw");
```

VB.Net

```
Dim user As New TdbUser(session)  
  
user.RealName = "My New User"  
user.LoginProcedure = "Public.Startup"  
user.Create("newuser", "newpw")
```

The following example assumes that the calling user is SYSTEM and attempts to create a new file manager. Note that if the calling user is not SYSTEM, the user will be created but will not be assigned any management privilege. If the calling user isn't a user manager, of course, the new user will not be created at all.

Java

```
TdbUser user = new TdbUser(session);
```

```
user.setRealName("My New DBA");
```

```
user.setIsFM(true);
```

```
user.create("new_fm", "newpw");
```

VB.Net

```
Dim user As New TdbUser(session)
```

```
user.RealName = "My New DBA"
```

```
user.IsFM = True
```

```
user.Create("new_fm", "newpw")
```

Modifying the properties of an existing user

In order to modify an existing user, simply retrieve the user's details from the server, modify locally and then store the modifications. To retrieve an existing user's properties, use the following method.

Method: TdbUser:Get

Type: void
Throws: TdbException

Java

```
void get(String username)
```

.Net

```
void Get(String username)
```

Retrieve the user's details from the server. This call will fail if the user is not owned by the calling user.

After making whatever changes are appropriate, commit the user back to the server using the Put method.

Method: TdbUser:Put

Type: void
Throws: TdbException

Java

```
void put()
```

.Net

```
void Put()
```

Store a previously retrieved user back to the server.

The following example shows modifying an existing user:

Java

```
TdbUser user = new TdbUser(session);

// Retrieve the user from the server
user.get("my_user");

// Modify the user's properties locally
user.setRealName("My User");
user.setLoginProcedure("Public.startup");

// Store the modifications back to the server
user.put();
```

VB.Net

```
Dim user As New TdbUser(session)

' Retrieve the user from the server
user.Get("my_user")

' Modify the user's properties locally
user.RealName = "My User"
user.LoginProcedure = "Public.startup"

' Store the modifications back to the server
user.Put()
```

As with creating new users, if the calling user is SYSTEM, the user being modified can be granted privileges by setting the appropriate property locally before committing the user object to the server.

Deleting existing users

To remove a user from the system, simply create a TdbUser object and invoke the Delete method.

Method: TdbUser:Delete

Type: void
Throws: TdbException

Java

```
void delete(String username)
void delete()
```

.Net

```
void Delete(String username)
void Delete()
```

Delete the named user from the server. This method is only usable by the user's manager. The second version of the method is only usable if the object was constructed using a valid Control object reference for the user, or if the user has already been retrieved from the server using the Get method.

Changing ownership

Users are normally owned (managed) by the user manager who creates them. However, over time that user may no longer be valid due to organizational changes, or the structure of TRIP management may change, requiring users to be moved between user managers.

To move one or more users between managers, use the ChangeMgr method.

Method: TdbUser:ChangeMgr

Type: void
Throws: TdbException

Java

```
void changeMgr(String newMgr, String oldMgr, String what)
void changeMgr(String newMgr, String oldMgr)
void changeMgr(String newMgr)
```

.Net

```
void ChangeMgr(String newMgr, String oldMgr, String what)
void ChangeMgr(String newMgr, String oldMgr)
void ChangeMgr(String newMgr)
```

Move one or more user objects between managers.

The various versions of this method reflect different operations that can be performed via the same underlying protocol.

- To move a single named user between two managers, use the first version of the method, specifying all parameters of the operation (null as "oldMgr" means the calling user).
- To move all user and group objects owned by one manager to another, use either the second or third versions of the method without having first retrieved a user from the server (null as "oldMgr" means the calling user).

- To move the current object, i.e. the user object after having retrieved the user's properties from the server, use either of the second or third versions of the method.

Note that in order to successfully specify an "oldMgr" that isn't the calling user, the calling user must be the SYSTEM user.

The following example shows various types of management change in action.

Java

```
// First, transfer a single user from me to another UM
TdbUser user1 = new TdbUser(session)
user1.changeMgr("someOtherUM", null, "theUserToMove");

// Again, transfer a specific user to another UM
TdbUser user2 = new TdbUser(session)
user2.get("someUser");
user2.changeMgr("someOtherUM");

// Finally transfer all remaining users from me to another UM
TdbUser user3 = new TdbUser(session)
user3.changeMgr("someOtherUM");
```

VB.Net

```
' First, transfer a single user from me to another UM
Dim user As New TdbUser(session)
user.ChangeMgr("someOtherUM", nothing, "theUserToMove")

' Again, transfer a specific user to another UM
Dim user2 As New TdbUser(session)
user2.Get("someUser")
user2.ChangeMgr("someOtherUM")

' Finally transfer all remaining users from me to another UM
Dim user3 As New TdbUser(session)
user3.ChangeMgr("someOtherUM")
```

16. Managing user groups

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

User groups are managed using the following class.

Class: TdbGroup

Derived from: TdbSerializableObject
Located in: users

Creating new groups

To create new groups, simply create a TdbGroup object and then invoke the Create method.

Method: TdbGroup.Create

Type: void
Throws: TdbException

Java

```
void create(String groupname)
```

.Net

```
void Create(String groupname)
```

Create a new user group with the provided name. This operation will fail if the calling user is not user manager, or if the group already exists.

Modifying group membership

To add or remove users to or from a user group, use the Add and Remove methods.

Method: TdbGroup.Add

Type: void
Throws: TdbException

Java

```
void add(String username, String groupname)
```

```
void add(String username)
```

.Net

```
void Add(String username, String groupname)
```

```
void Add(String username)
```

The second version of this method is only usable if the TdbGroup object was constructed using the group's name or a valid Control object reference for the group in question.

Method: TdbGroup:Remove

Type: void
Throws: TdbException

Java

```
void remove(String username, String groupname)
void remove(String username)
```

.Net

```
void Remove(String username, String groupname)
void Remove(String username)
```

The second version of this method is only usable if the TdbGroup object was constructed using the group's name or a valid Control object reference for the group in question.

Deleting existing groups

To delete a group from the system, use the Delete method.

Method: TdbGroup:Delete

Type: void
Throws: TdbException

Java

```
void delete(String groupname)
void delete()
```

.Net

```
void Delete(String groupname)
void Delete()
```

Delete the named user group from the system. In order to use the second version of the method, the object must have been constructed using the group's name or a valid Control object reference for the group.

Changing ownership

Groups are normally owned (managed) by the user manager who creates them. However, over time that user may no longer be valid due to organizational changes, or the structure of TRIP management may change, requiring users to be moved between user managers.

To move one or more groups between managers, use the ChangeMgr method.

Method: TdbGroup:ChangeMgr

Type: void
Throws: TdbException

Java

```
void changeMgr(String newMgr, String oldMgr, String what)
void changeMgr(String newMgr, String oldMgr)
void changeMgr(String newMgr)
```

.Net

```
void ChangeMgr(String newMgr, String oldMgr, String what)
void ChangeMgr(String newMgr, String oldMgr)
void ChangeMgr(String newMgr)
```

Move one or more user objects between managers.

The various versions of this method reflect different operations that can be performed via the same underlying protocol.

- To move a single named group between two managers, use the first version of the method, specifying all parameters of the operation (null as “oldMgr” means the calling user).
- To move all user and group objects owned by one manager to another, use either the second or third versions of the method without having constructed the group object using its name.
- To move the current object, i.e. the group object having been constructed using the group’s name, use either of the second or third versions of the method.

Note that in order to successfully specify an “oldMgr” that isn’t the calling user, the calling user must be the SYSTEM user.

The following example shows various types of management change in action.

Java

```
// First, transfer a single user from me to another UM
TdbGroup group1 = new TdbGroup(session)
group1.changeMgr("someOtherUM", null, "theGroupToMove");

// Again transfer a specific group to another UM
TdbGroup group2 = new TdbGroup(session, "someGroup");
group2.changeMgr("someOtherUM");

// Finally transfer all users and groups from me to another UM
TdbGroup group3 = new TdbGroup(session)
group3.changeMgr("someOtherUM");
```

VB.Net

```
' First, transfer a single group from me to another UM
Dim group As New TdbGroup(session)
group.ChangeMgr("someOtherUM", nothing, "theGroupToMove")

' Again transfer a specific group to another UM
Dim group2 As New TdbGroup(session, "someGroup")
group2.ChangeMgr("someOtherUM")

' Finally transfer all users and groups from me to another UM
Dim group3 As New TdbGroup(session)
group3.ChangeMgr("someOtherUM")
```


17. Managing stored procedures

Note that this chapter relates to physical sessions only. Any attempt to use these classes with a TRIPgrid session will result in an UNUSABLE_SESSION exception being thrown.

Stored procedures are managed using the TdbProcedure class.

Class: TdbProcedure

Derived from: TdbSerializableObject
Located in: users

This class provides all management applications for stored procedures, but does not support execution of such procedures, for which the TdbCclCommand class is intended.

Creating new procedures

To create a new stored procedure, simply create an instance of TdbProcedure, set its properties as required, and use the Create method to commit the new procedure to the server.

Method: TdbProcedure.Create

Type: void
Throws: TdbException

Java

```
void create(String owner, String name, String comment)
```

.Net

```
void Create(String owner, String name, String comment)
```

Create a new stored procedure. If the owner parameter is null or of zero length, the owner is set to the calling user, otherwise the user or group named will be defined as the owner of the new procedure.

The content of the procedure is retrieved and/or established using the Content collection.

Collection: TdbProcedure.Content

Type: List of String
Access: Read, Write

Java

```
List<String> content()  
void setContent(Collection<String>)
```

.Net

```
ICollection Content { get; set; }
```

The content of the procedure is reflected as a snapshot, not a live collection. That is, applications should retrieve the content, do with it as they may, and then update the content explicitly.

Note that the TdbProcedure class is also natively enumerable and that the iterator generated using that access means is also backed by the Content collection, i.e. a snapshot and not the live content.

The following example shows how to create a new procedure.

Java

```
TdbProcedure proc = new TdbProcedure(session)

// Set the content of the new procedure
Vector<String> content = new Vector<String>;
content.add("base alice");
content.add("find mad hatter");
proc.setContent(content);

// Commit the procedure to the server
proc.create(null, "my_new_proc", "some comment");
```

VB.Net

```
Dim proc As New TdbProcedure(session)

' Set the content of the new procedure
Dim content As New ArrayList()
content.Add("base alice")
content.Add("find mad hatter")
proc.Content = content

' Commit the new procedure to the server
proc.Create(Nothing, "my_new_proc", "some comment")
```

Create procedure based on a search

The `SaveSearch` method of the `TdbProcedure` class, introduced with TRIP 7.2, provides a means by which a procedure can be created based on a search more conveniently.

Method: TdbProcedure:SaveSearch

Type: void
Throws: TdbException

Java

```
void saveSearch(int searchSetNumber, String owner,  
                String name, String comment)
```

.Net

```
void SaveSearch(int searchSetNumber, String owner,  
                String name, String comment)
```

Create a new stored procedure based on the CCL command that generated the specified search set. Any other CCL commands that the main CCL command depends upon will be included in the procedure. If the owner parameter is null or of zero length, the owner is set to the calling user, otherwise the user or group named will be defined as the owner of the new procedure.

NOTE: This procedure must be executed using the FIND SAVE command.

Modifying existing procedures

In order to modify an existing procedure, simply retrieve the procedure from the server using the Get method, modify it locally and then store the resulting updated procedure to the server using the Put method.

Method: TdbProcedure:Get

Type: void
Throws: TdbException

Java

```
void get(String owner, String name)  
void get()
```

.Net

```
void Get(String owner, String name)  
void Get()
```

Retrieve the named procedure from the server. In the first version of the method, set the owner parameter to null for the calling user. The second version of the method is only available if the TdbProcedure object was constructed using a valid Control object reference for the procedure in question.

Method: TdbProcedure:Put

Type: void
Throws: TdbException

Java

```
void put()
```

.Net

```
void Put()
```

Store an updated procedure back to the server. Note that if the procedure was not first retrieved using the Get method, the Put method will attempt to create a new procedure potentially causing an exception to be thrown by the server if the procedure already exists.

The following example shows how to retrieve, modify and then store an updated group procedure.



Java

```
TdbProcedure proc = new TdbProcedure(session);

// Retrieve the procedure from the server
proc.get("group_1", "grp_proc");

// Show the procedure to the end user
for( String line : proc )
    System.out.println(line);

// Modify the procedure in some way
proc.putComment("Some new comment");

// Now store the updated procedure back to the server
proc.put();
```

VB.Net

```
Dim proc As New TdbProcedure(session)

' Retrieve the procedure from the server
proc.Get("group_1", "grp_proc");

' Show the procedure to the end user
For Each line As String In proc
    myEditBox.Text += line + Environment.NewLine
Next

' Modify the procedure in some way
proc.Comment = "Some new comment"

' Now store the updated procedure back to the server
proc.Put()
```

Deleting existing procedures

To delete a procedure, simply create a `TdbProcedure` object and invoke the `Delete` method.

Method: `TdbProcedure.Delete`

Type: `void`
Throws: `TdbException`

Java

```
void delete(String owner, String name)
void delete()
```

.Net

```
void Delete(String owner, String name)
void Delete()
```

Delete the named procedure from the server. Applications can pass a null reference for the owner to signify the calling user. Note that the second version of the method is only usable if the `TdbProcedure` object was constructed using a valid `Control` object reference for the procedure in question.

18. Connection pooling

In certain application contexts, it can be very valuable in terms of performance to reuse a single session many times rather than starting a new session for each request. This is particularly important when the startup cost for a session, for example in the face of a complex startup procedure, is very high compared to the amount of time that the session is expected to live.

For applications that require a large number of users to access TRIP via a common login (i.e. the same username for all, or the majority of, sessions), a connection pool can provide a convenient means for managing this startup time issue.

For the sake of illustration, assume that a particular application has a security environment such that the manager of the data authenticates using an account called **MANAGER**, whilst all read-only users of that application access data using an account called **USER**. In this case, all of those query users can be managed via a single connection pool, whilst the manager would access the system via a normal session login, as described in chapter 3.

A connection pool simply manages an extensible set of sessions (of either TRIPnet or Web types only). Sessions in a connection pool are acquired and released to the pool, rather than being created and logged out, as is the case with a normal session.

Connection pools are created using a specialization of the connection pool class:

Class: TdbConnectionPool

Derived from: object
Located in: pool

Note that this is an abstract class and therefore cannot be instantiated directly. Instead create an instance of one of the following specialized types.

Class: TdbTripNetConnectionPool

Derived from: TdbConnectionPool
Located in: pool

This class creates and manages a pool of TRIPnet sessions.

Class: TdbWebConnectionPool

Derived from: TdbConnectionPool
Located in: pool

This class creates and manages a pool of XML/HTTP web sessions.

Once the connection pool is created, sessions are acquired from the pool using the **Acquire** method. When the application is done with the session, the **Logout** method can be used on the session to return that session to the pool.

Constructor: TdbTripNetConnectionPool*Java*

```
TdbTripNetConnectionPool(String server, int port,
                          String username, String password,
                          TdbLanguage lang, int stepping)
```

.Net

```
TdbTripNetConnectionPool(String server, int port,
                          String username, String password,
                          TdbLanguage lang, int stepping)
```

Create a connection pool with sessions that uses a TRIPnet network connection to the server.

The final constructor argument is a “stepping” value that details the number of connections that will be created at any one time. Note that sessions are not authenticated, and hence do not take the cost of the login process, whatever that might be for the application in question, until the session is acquired for the first time from the connection pool.

Method: TdbConnectionPool:Acquire

Type: TdbPooledSession
Throws: TdbException

Java

```
TdbPooledSession acquire();
```

.Net

```
TdbPooledSession Acquire();
```

Returns a session from the connection pool. If the connection pool is full, this method will throw a TdbException exception with the Code property (in TRIPjxp, use getCode() to get the value) set to UNUSABLE_SESSION. If this is the case, the application should wait a bit and try calling acquire() again. For any other Code value, the application should fail the acquire attempt.

Method: TdbConnectionPool:Close

Type: TdbPooledSession
Throws: n/a

Java

```
void close();
```

.Net

```
void Close();
```


Shuts down the connection pool.

Even though a connection pool will eventually be destroyed anyway by the garbage collector, the pool is associated with a number of resources on both the client machine and the server machine. It is therefore strongly recommended that your code explicitly calls the `Close()` method when it is done with the pool.

The following example shows how to create a connection pool, set up purging of idle sessions, how to acquire sessions from the pool, and how to return sessions to the pool for reuse.

Java

```
// Create a connection pool of TRIPnet sessions – all sessions
// that are acquired from this pool will be authenticated as
// the system manager
TdbConnectionPool pool;
pool = new TdbTripNetConnectionPool("myserver", 23457
                                     "system", "z",
                                     TdbLanguage.English, 5);

// Set up how idle sessions are to be handled. Allow
// idle sessions to time out after 55 seconds, the purge
// routine is run every 60 seconds, and 5 idle sessions may
// remain in the pool.
pool.setIdleTimeout(55);
pool.setPurgeInterval(60);
pool.setMinIdle(5);

// Acquire a session from the pool – if necessary this will
// perform a full login, otherwise the session will simply be
// initialized for reuse if it has already been authenticated.
// NOTE: This acquire-loop omits things like sleep (optional)
// and a timeout for when the app should stop trying.
TdbSession session = null;
do
{
```

```
try
{
    session = pool.acquire();
}
catch (TdbException tdbx)
{
    if (tdbx.getCode() != TdbException.UNUSABLE_SESSION)
    {
        // Error other than 'no sessions available'.
        // Abort the acquire attempt!
        throw new Exception("Acquire failed.", tdbx);
    }
}

while (null == session);

// The session can now be used exactly like a normal
// TRIPnet session
TdbDatabaseList dblist = new TdbDatabaseList(session);
// ... etc. ...

// Now return the session to the connection pool
session.logout();

// At application shutdown, or when the pool is no longer
// needed, you MUST close the pool!
pool.close();
```

VB.Net

```
Dim pool As TdbConnectionPool = Nothing
Dim dblist as TdbDatabaseList = Nothing
Dim session As TdbSession = Nothing
```

```

' Create a connection pool of TRIPnet sessions – all sessions
' that are acquired from this pool will be authenticated as
' the system manager
pool = New TdbTripNetConnectionPool("myserver", 23457,
                                     "system", "z",
                                     TdbLanguage.English, 5)

' Set up how idle sessions are to be handled. Allow
' idle sessions to time out after 55 seconds, the purge
' routine is run every 60 seconds, and 5 idle sessions may
' remain in the pool.
pool.IdleTimeout = 55
pool.PurgeInterval = 60
pool.MinIdle = 5

' Acquire a session from the pool – if necessary this will
' perform a full login, otherwise the session will simply be
' initialized for reuse if it has already been authenticated
' NOTE: This acquire-loop omits things like sleep (optional)
' and a timeout for when the app should stop trying.
while (session Is Nothing)
    Try
        session = pool.Acquire()
    Catch ex As TdbException
        If ex.Code <> TdbException.UNUSABLE_SESSION Then
            ' Other error than "no session is available now".
            ' Abort the attempt.
            Throw New Exception("Acquire failed.", ex)
        End If
    End Try
End while

' The session can now be used exactly like a normal
' TRIPnet session

```

```
dblist = New TdbDatabaseList(session)
```

```
' Now return the session to the connection pool  
session.Logout()
```

```
' At application shutdown, or when the pool is no longer  
' needed, you MUST close the pool!  
pool.Close()
```



19. Interaction with TRIPcof

TRIPcof implement functionality for extraction of text and properties from the most popular document file formats (office documents, PDF, etc), and conversion of said formats to HTML, optionally with search hit highlighting.

This functionality area was previously handled by the add-on products TRIPview and TRIPview-C. These are no longer available, although the APIs described in this chapter also applies to them.

API overview

Property **TdbStringField:ExtractionTarget**

Use the ExtractionTarget property of the field object to define the name of the TExt field that will receive any textual content found.

NOTE: Extracting text this way is NOT supported with TRIPcof. This method is only provided for backward compatibility purposes.

This property is DEPRECATED from version 3.0 of TRIPnpx and TRIPjxp.

Property: **TdbStringField:ExtractionTarget**

Type: String
Access: Read, write

Java

```
String getExtractionTarget();  
void setExtractionTarget(String name);
```

.Net

```
String ExtractionTarget { get; set; }
```

If this property is set before the record is committed, the commit process will include a call to TRIPview (assuming the server is suitably licensed for that operation). The original binary content will be stored in the related SString field of the record, and any text extracted from the SString by TRIPview will be stored in the TExt field named by this property.

Property **TdbTextField:TextExtractionInfo**

For text extraction operations with version 1.2 or later of TRIPnXP or TRIPjXP, use of the TextExtractionInfo property on the TdbTextField class is recommended instead of using the ExtractionTarget property on the TdbStringField.

Property: **TdbTextField:TextExtractionInfo**

Type: TdbTextExtractionInfo
Access: Read

Java

```
TdbTextExtractionInfo getTextExtractionInfo();
```

.NET

```
TdbTextExtractionInfo TextExtractionInfo { get; }
```

This property returns a TdbTextExtractionInfo object that is used to control and enable text extraction operations.

Class **TdbTextExtractionInfo**

This class is a container for text extraction directives.

Class: **TdbTextExtractionInfo**

Derived from: Object
Located in: data

This class has no public constructor. Instances can be accessed via the TdbTextField.TextExtractionInfo property described above.

Property: **TdbTextExtractionInfo:BinaryCopyField**

Type: String
Access: Read, Write

Java

```
String getBinaryCopyField();  
void setBinaryCopyField(String fieldName);
```

.NET

```
String BinaryCopyField { get; set; }
```

Gets and sets the name of the string field that receives a copy of the document data, and/or contains the document data to extract text from.

If a string field is specified in this property and neither the FileName nor the Stream property is specified, text extraction will be performed from the data already stored in the specified string field. Note that this is only supported for updates of existing records.

Property: TdbTextExtractionInfo:ClientSide

Type: bool
Access: Read, Write

Java

```
boolean getClientSide();  
void setClientSide(boolean enable);
```

.NET

```
bool ClientSide { get; set; }
```

Determines if the text extraction should be performed on the server or on the client-side. Default is false, which results in server-side text extraction.

Client-side text extraction requires a local installation of TRIPcof. Users who owns a license of TRIPview-C and have such an installation on the client-side, will also be able to use this functionality.

If TRIPcof is used:

- For TRIPjxp, the TRIPcof installation directory must be specified as either a system property with the name TRIPCOF_HOME, or an environment variable with the name TRIPCOF_HOME.
- TRIPnxp reads the TRIPCOF_HOME information from the Windows registry.

If TRIPview-C is used:

- For TRIPjxp, the TRIPview-C installation directory must be specified as either a system property with the name TRIPVIEW_HOME, or an environment variable with the name TRIPVIEW_HOME.
- TRIPnxp reads the TRIPVIEW_HOME information from the Windows registry.

If the text extraction is performed on the client-side, it is fully performed before commit of the record. The data transmitted to the server for the associated text field will be the extracted text. No further processing is done on the server.

Property: TdbTextExtractionInfo:ExtractText

Type: bool
Access: Read, Write

Java

```
boolean getExtractText();  
void setExtractText(boolean enable);
```

.NET

```
bool ExtractText { get; set; }
```

This property determines if a text extraction operation is to be performed. If set to true, text will be extracted from the specified file data and stored in the TEXT field that the current TdbTextExtractionInfo instance is associated with.

When ExtractText is set to true, you must also in addition at least set FileName property. To store a copy of the file, provide the name of a STRING field to the BinaryCopyField property. The file data can also be provided via a stream (use the Stream property), which is an option if the file does not exist physically on the local machine.

To extract text during update of a record from data already stored in a STRING field, set the BinaryCopyField to the STRING field name and the ExtractFromStored property to true.

Property: TdbTextExtractionInfo:ExtractFromStored

Type: String
Access: Read, Write

Java

```
boolean getExtractFromStored();  
void setExtractFromStored(boolean enable);
```

.NET

```
bool ExtractFromStored { get; set; }
```

This property determines if the text extraction should be performed from the value already stored in the STRING field defined by the property BinaryCopyField.

Set this property to true if you wish to perform a text extraction on data previously added to a STRING field, but do not intend or wish to supply the value again. The text extraction specification should not be assigned a value to the Stream property, but the FileName property is still a good idea to set in order to inform TRIPcof of the name (and type) of the file.

This property is false by default.

Property: TdbTextExtractionInfo:FileName

Type: String
Access: Read, Write

Java

```
String getFileName()  
void setFileName(String fileName);
```

.NET

```
String FileName { get; set; }
```

The file name is a valuable aid to the text extractor in helping to determine the type of the file, if it cannot be determined by any other means. It is therefore recommended that the file name is specified even if the data to extract from is specified using the Stream property.

If FileName is specified and refers to an existing local file and the Stream property is null, text extraction will be performed from the named file. This is the preferred choice if the application is running on the TRIP server machine.

Property: TdbTextExtractionInfo:PropertyNameField

Type: String
Access: Read, Write

Java

```
String getPropertyNameField()  
void setPropertyNameField(String name);
```

.NET

```
String PropertyNameField { get; set; }
```

A phrase field to receive document property names during text extraction.

This property is used together with the PropertyValueField property. Either these two properties are both set, or both are not set. A text extraction operation cannot be performed if one of these properties are set and the other is not.

Property: TdbTextExtractionInfo:PropertyValueField

Type: String
Access: Read, Write

Java

```
String getPropertyValueField()  
void setPropertyValueField(String name);
```

.NET

```
String PropertyValueField { get; set; }
```

A phrase field to receive document property values during text extraction.

This property is used together with the `PropertyNameField` property. Either these two properties are both set, or both are not set. A text extraction operation cannot be performed if one of these properties are set and the other is not.

Property: `TdbTextExtractionInfo:Stream`

Type: `InputStream (java), Stream (.NET)`
Access: `Read, Write`

Java

```
java.io.InputStream getStream()
void setStream(java.io.InputStream is);
```

.NET

```
System.IO.Stream Stream { get; set; }
```

Stream for document data to extract text from and optionally store a copy of. If the application is not running on the same machine with the server, this property should be used to provide document data. Note that the `FileName` property should still be set if a file name is known, since this helps TRIPcof to determine the type of the file if the file type cannot be determined by other means.

Property: `TdbTextExtractionInfo:StringField`

Type: `TdbStringField`
Access: `Read, Write`

Java

```
TdbStringField getStringField();
void setStringField(TdbStringField field)
```

.NET

```
TdbStringField StringField { get; set; }
```

If this property is assigned a non-null value, text extraction will be performed from any value assigned to the provided `TdbStringField` instance.

When this property is assigned a non-null value, the following apply:

- the value of the `BinaryCopyField` is assigned the name of the assigned string field automatically.
- the property `Stream` will be ignored during text extraction

If text extraction is to be performed from a value already stored in a string field, it is better not to include the `TdbStringField` instance at all in the `TdbRecord` object used for commit, and to set the `ExtractFromStored` property to true instead.

Property: TdbTextExtractionInfo:TextField

Type: String
Access: Read

Java

```
String getTextField();
```

.NET

```
String TextField { get; }
```

Returns the name of the text field in which the extracted text is to be stored. This is the name of the TdbTextField object that the current TdbTextExtractionInfo instance is obtained from.



Method `TdbStringField.Convert`

This method was added to version 3.0 of TRIPnxp and TRIPjxp for the purpose of performing client-side HTML conversion using a local installation of TRIPcof. Users of TRIPview-C will also be able to use this method.

A good practice with regard to HTML conversion is to perform it as close to the consumer as possible. This normally means on an application server running a web application based on TRIPjxp or TRIPnxp.

Method: `TdbStringField.Convert`

Type: `void`
Throws: `TdbException`

Java

```
void convert(TdbRenditionType rendition,  
            String originalFilename,  
            String outputDirectory,  
            String outputFilename,  
            String preferredAdapter,  
            TdbTextField highlightSourceField,  
            boolean refreshHitLocations)
```

.Net

```
void Convert(TdbRenditionType rendition,  
            String originalFilename,  
            String outputDirectory,  
            String outputFilename,  
            String preferredAdapter,  
            TdbTextField highlightSourceField,  
            bool refreshHitLocations)
```

The use of this method requires a local installation of TRIPcof or TRIPview-C.

If TRIPcof is used:

- For TRIPjxp, the TRIPcof installation directory must be specified as either a system property with the name `TRIPCOF_HOME`, or an environment variable with the name `TRIPCOF_HOME`.
- TRIPnxp reads the `TRIPCOF_HOME` information from the Windows registry.

If TRIPview-C is used:

- For TRIPjxp, the TRIPview-C installation directory must be specified as either a system property with the name `TRIPVIEW_HOME`, or an environment variable with the name `TRIPVIEW_HOME`.
- TRIPnxp reads the `TRIPVIEW_HOME` information from the Windows registry.

In order to perform client-side HTML conversion with this method, the string field value to convert must first be fetched from the server without a rendition specification (use a normal `TdbFieldTemplate` or a `TdbRendition` with `DefaultRendition` as the rendition type). When the string value has been fetched, call this method with a HTML-specific rendition type.

Note that the rendition type BasicHTML may generate any number of graphic files for the images in the document. These files will be put in the same directory with the generated HTML file. Be sure to specify the filename argument so that it is easy to clean up these files when they are no longer needed.

Hit highlighting is supported when the associated search set has been created by a FIND or FUZZ CCL order to either the TdbCclCommand class or the TdbSearch class. The TdbRecordSet class is only supported if a pre-existing search order is supplied to it. Using the query capability of the TdbRecordSet class itself is not supported because such search sets are automatically deleted before the data is returned to the client.

In order to generate an HTML file where search hits are highlighted, supply the text field object to the *highlightSourceField* parameter. This text field must contain the extracted document text.

The *refreshHitLocations* parameter should be set to true when you are using HTML conversion with TRIPview-C. Assigning this parameter to false when TRIPview-C is used may result in incorrect offset values to hit words. This is because fill characters produced during text extraction may differ from version to version. Setting this parameter to true results in extra network I/O.

To produce an HTML document without hit highlighting, pass null to the *highlightSourceField* parameter.

For server-side conversion, add a TdbRendition or TdbHighlightRendition instance as field template to the retrieval template instead of a normal TdbFieldTemplate. Then use the file CopyToFile to store the rendered data to a local file, or access it using the Rendition property.

Requesting an HTML conversion with hit highlighting results in network I/O.

Extracting text from a stream with server-side processing

This procedure requires version 1.2 or later of TRIPnXP or TRIPjXP. TRIPcof or TRIPview-C must be installed on the server.

Java

```
TdbDatabaseDesign db;
TdbRecord          rec;
TdbTextField        tfield;
TdbTextExtractionInfo txi;

// Retrieve database design, create new record
db = new TdbDatabaseDesign(session);
db.get("VIEWDEMO2");
record = new TdbRecord(session, db, false);
head = record.getHead();

// Create field object
TdbFieldDesign fdes = db.getFieldByName("FILE_TEXT")
tfield = (TdbTextField) head.createField(fdes);

// Assign extraction data
txi = tfield.getTextExtractionInfo();
txi.setBinaryCopyField("FILE_BLOB")
txi.setExtractText(true);
txi.setFileName("EXAMPLE.DOC");
txi.setStream(streamToDocumentData); // Provided by app
txi.setPropertyNameField("PROP_NAME");
txi.setPropertyValueField("PROP_VALUE");

// Store new record to database
record.commit();
```

VB.NET

```
' Retrieve database design, create new record
Dim db As New TdbDatabaseDesign(session)
db.Get("VIEWDEMO2")
Dim record As New TdbRecord(session, db, false)
```

```
Dim head As TdbComponent = record.Head

' Create field object
Dim field As TdbTextField
field = _
    DirectCast(head.CreateField(db("FILE_TEXT")), TdbTextFiel)

' Set text extraction options
With field.TextExtractionInfo
    .ExtractText = true
    .BinaryCopyField = "FILE_BLOB"
    .FileName = "EXAMPLE.DOC"
    .Stream = streamToDocumentData ' Provided by application
    .PropertyNameField = "PROP_NAME"
    .PropertyValueField = "PROP_VALUE"
End With

' Store new record to database
record.Commit()
```

Extracting text from a previously stored file during update

This procedure requires TRIPNxp/TRIPjxp 2.1 version 2.1-8 or later, or TRIPNxp/TRIPjxp 3.0 version 3.0-2 or later.

Java

```
TdbDatabaseDesign db;

TdbRecord          rec;

TdbTextField        tfield;

TdbTextExtractionInfo txi;


// Retrieve database design, create new record object
db = new TdbDatabaseDesign(session);
db.get("VIEWDEMO2");
record = new TdbRecord(session, db, true);


// Set ID of record to update - 'rid' provided by application
record.setRecordId(rid);


head = record.getHead();


// Create field object
TdbFieldDesign fdes = db.getFieldByName("FILE_TEXT")
tfield = (TdbTextField) head.createField(fdes);


// Assign extraction data
txi = tfield.getTextExtractionInfo();
txi.setBinaryCopyField("FILE_BLOB")
txi.setExtractText(true);
txi.setExtractFromStored(true);
txi.setFileName("EXAMPLE.DOC");
txi.setPropertyNameField("PROP_NAME");
txi.setPropertyValueField("PROP_VALUE");


// Perform update of record
record.commit();
```


VB.NET

```
' Retrieve database design, create new record object
Dim db As New TdbDatabaseDesign(session)
db.Get("VIEWDEMO2")
Dim record As New TdbRecord(session, db, false)

' Set ID of record to update - 'rid' provided by application
record.RecordId = rid

Dim head As TdbComponent = record.Head

' Create field object
Dim field As TdbTextField
field = _
    DirectCast(head.CreateField(db("FILE_TEXT")), TdbTextFiel)

' Set text extraction options
With field.TextExtractionInfo
    .ExtractText = true
    .ExtractFromStored = true
    .BinaryCopyField = "FILE_BLOB"
    .FileName = "EXAMPLE.DOC"
    .PropertyNameField = "PROP_NAME"
    .PropertyValueField = "PROP_VALUE"
End With

' Store new record to database
record.Commit()
```

HTML conversion

This procedure requires version 1.2 or later of TRIPnpx or TRIPjxp, and TRIPcof or TRIPview-C.

Rendering the contents of a STRING field as HTML is a procedure that makes use of the TdbRendition class, which is a specialization of the TdbFieldTemplate class. How to produce HTML renditions has been described elsewhere in this document, but here is a short example:

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
TdbRecord tmp1 = new TdbRecord(session);
String fileDataField = "FILE_BLOB";
String fileNameField = "FILE_NAME";

tmp1.addToTemplate(new TdbRendition(fileDataField,
    fileNameField, TdbRenditionType.BasicHTML));
rs.setRetrievalTemplate(tmp1);

// Set other properties
// ...

rs.get();
```

VB.NET

```
Dim rs as new TdbRecordSet(session)
Dim tmp1 as new TdbRecord(session)
Dim fileDataField As String = "FILE_BLOB"
Dim fileNameField As String = "FILE_NAME"

tmp1.AddToTemplate(new TdbRendition(fileDataField, _
    fileNameField, TdbRenditionType.BasicHTML))
rs.RetrievalTemplate = tmp1

' Set other properties
' ...

rs.Get()
```

This will produce a result set where the contents of the STRING field (in this example called "FILE_BLOB"), is returned rendered as HTML.

Note that the a field containing the name of the file is required. Without this information, TRIPcof may not be able to create an HTML rendition of the file. This version of the TdbRendition constructor was introduced in version 3.0 of TRIPnXP and TRIPjXP.

HTML conversion with HTML highlighting

With TRIPjxp and TRIPnxp version 1.2 it is possible to convert a STRING field to HTML with search hit highlighting using the TdbHighlightRendition class.

Class: TdbHighlightRendition

Derived from: TdbRendition
Located in: data

This specialization of the rendition field template is identical to its base class in all aspects except that it specifies that search hit highlighting is to applied to the output using the hit vector from a TEXT field.

Java

```
TdbHighlightRendition(String stringFieldName,  
                      String textFieldName,  
                      TdbRenditionType type)  
  
TdbHighlightRendition(String stringFieldName,  
                      String textFieldName,  
                      String filenameFieldName,  
                      TdbRenditionType type)
```

.Net

```
TdbHighlightRendition(String stringFieldName,  
                      String textFieldName,  
                      TdbRenditionType type)  
  
TdbHighlightRendition(String stringFieldName,  
                      String textFieldName,  
                      String filenameFieldName,  
                      TdbRenditionType type)
```

The TRIPcof-associated rendition types supported for TdbHighlightRendition via the TdbRenditionType enumeration are HTML and Mime-Encoded HTML (MHTML file format, including all embedded graphics).

The first version of the constructor is suitable only for use with TRIPxml to retrieve stored XML documents with hit markup applied.

The second version of the constructor is recommended for use with TRIPcof and TRIPview-C for HTML conversion. This constructor variant was added in version 3.0 of TRIPnxp and TRIPjxp.

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setQuery("FIND FILETEXT=TRIPjxp");
String fileDataField = "FILE_BLOB";
String fileTextField = "FILE_TEXT";
String fileNameField = "FILE_NAME";

TdbRecord tmp1 = new TdbRecord(session);
tmp1.addToTemplate(new TdbHighlightRendition(fileDataField,
    fileTextField, fileNameField, TdbRenditionType.BasicHTML));
rs.setRetrievalTemplate(tmp1);

// Set other properties
// ...

rs.get();
```

VB.NET

```
Dim rs as new TdbRecordSet(session)
String fileDataField = "FILE_BLOB";
String fileTextField = "FILE_TEXT";
String fileNameField = "FILE_NAME";

rs.Query = "FIND FILETEXT=TRIPnxp"

Dim tmp1 as new TdbRecord(session)
tmp1.AddToTemplate(new TdbHighlightRendition(fileDataField,
    fileTextField, fileNameField, TdbRenditionType.BasicHTML))
rs.RetrievalTemplate = tmp1

' Set other properties
' ...

rs.Get()
```

Example of client-side HTML conversion with highlighting

This shows how to use the Convert method in the TdbStringField class to perform client-side HTML conversion with highlighting using the LibreOffice file filter adapter in a local TRIPcof installation.

See also the TRIPnxp example program CofClient and the TRIPjxp example programs CofConvert and CofExtract.

Java

```
// Use TdbSearch (not TdbRecordSet) to search
TdbSearch search = new TdbSearch(session);
search.execute("BASE COFDEMO");
search.execute("FIND FILE_TEXT=TRIPcof");

TdbRecord rec = search.getLastSearchSet().getRecord(0);
TdbComponent head = rec.getHead();
TdbStringField sf = (TdbStringField) head.getField("FILE_BLOB");
TdbPhraseField nf = (TdbPhraseField) head.getField("FILE_NAME");

String outputDirectory = "C:\\Data\\Output";
String outputFile = "example.html";

fld.convert(TdbRenditionType.BasicHTML, nf.getValue(0),
            outputDirectory, outputFile, "libreoffice",
            "FILE_TEXT", true);
```

Text Analysis

Version 3.0-1 of TRIPnpx and TRIPjxp added support for TRIPcof-based analysis of text stored in TRIP records in order to extract keywords and metadata. This analysis requires that TRIPcof has been configured with an NLP analysis adapter.

Property: TdbRecord:NlpInfo

Type: TdbNlpInfo
Access: Read

Java

```
TdbNlpInfo getNlpInfo();
```

.NET

```
TdbNlpInfo NlpInfo { get; }
```

This property returns a TdbNlpInfo object that is used to control and enable text analysis operations.

Class TdbNlpInfo

This class is a container for NLP analysis directives.

Class: TdbNlpInfo

Derived from: Object
Located in: data

This class has no public constructor. Instances can be accessed via the TdbRecord.NlpInfo property described above.

Property: TdbNlpInfo:IsEnabled

Type: Boolean
Access: Read

Java

```
boolean isEnabled();
```

.NET

```
bool IsEnabled { get; }
```

Returns true if TRIPcof NLP interaction is enabled and false otherwise. TRIPcof NLP analysis is enabled if either keywords or meta-data extraction has been enabled. See properties ExtractKeywords and ExtractMetadata.

Property: TdbNlpInfo:ExtractKeywords

Type: Boolean
Access: Read

Java

```
boolean getExtractKeywords();
void setExtractKeywords(boolean enable);
```

.NET

```
bool ExtractKeywords { get; set; }
```

Enables or disables the extraction of keywords. If enabled, the KeywordField property must also be set.

Property: TdbNlpInfo:ExtractMetadata

Type: Boolean
Access: Read, Write

Java

```
boolean getExtractMetadata();
void setExtractMetadata(boolean enable);
```

.NET

```
bool ExtractMetadata { get; set; }
```

Enables or disables the extraction of metadata. If enabled, the property FieldMapping must also be set.

Method: TdbNlpInfo:SetInputLanguage

Type: void
Throws: N/A

Java

```
void setInputLanguage(int lang);
void setInputLanguage(String lang);
void setInputLanguage(TdbLanguage lang);
```

.Net

```
void SetInputLanguage(int lang);
void SetInputLanguage(String lang);
void SetInputLanguage(TdbLanguage lang);
```

Defines the input language of the text in the record to be analyzed. Input language should be declared if the database design does not include a natural language specification.

Depending on the NLP adapter used with TRIPcof, specifying the input language may not be necessary if the adapter is capable of determining the language of the text to be analyzed. However, as such extra analysis adds some (minor)

overhead to the total processing time, so it is better to specify the language explicitly if it is known beforehand.

Property: TdbNlpInfo:FieldMapping

Type: FieldMappingType
Access: Read, Write

Java

```
FieldMappingType getFieldMapping();  
void setFieldMapping(FieldMappingType type);
```

.NET

```
FieldMappingType FieldMapping { get; set; }
```

Get or set the field name mapping mode that controls how extracted meta-data is stored in the record. The FieldMappingType type is an enum declared in the TdbNlpInfo class. Its values are:

None	No metadata analysis output used.
Fields	Each metadata category correspond to a separate field in the database.
Tuples	The database has two fields that form a name/value pair tuple list that can store any category name and value found during analysis.

Property: TdbNlpInfo:PreferredAdapter

Type: String
Access: Read, Write

Java

```
String getPreferredAdapter();  
void setPreferredAdapter(String adapter);
```

.NET

```
String PreferredAdapter { get; set; }
```

Assign the ID of the adapter that must be used to perform the analysis. If not set (or set to an empty string), any suitable one installed on the server will be used.

Property: TdbNlpInfo:Fallback

Type: Boolean
Access: Read, Write

Java

```
boolean getFallback();  
void setFallback(boolean enable);
```

.NET

```
bool Fallback { get; set; }
```

If fallback is enabled and processing fails using one adapter, TRIPcof will attempt to use other NLP adapters if any are available.

Property: TdbNlpInfo:KeywordField

Type: String
Access: Read, Write
Throws: TdbException

Java

```
String getKeywordField();  
void setKeywordField(String fieldname);
```

.NET

```
String keywordField { get; set; }
```

Get or set the name of the PHRASE field to receive extracted keywords. To extract keywords, you must also set the ExtractKeywords property to true.

Method: TdbNlpInfo:SetMetadataTupleFields

Type: void
Throws: TdbException

Java

```
void setMetadataTupleFields(String namefield,  
                             String valuefield);
```

.Net

```
void setMetadataTupleFields(String namefield,  
                             String valuefield);
```

Set the names of the PHRASE fields to receive extracted meta-data. The field mapping must be set to TdbNlpinfo.FieldMappingType.Tuples if these field names are to be considered by TRIPcof.

Property: TdbNlpInfo:MetadataNameField

Type: String
Access: Read

Java

```
String getMetadataNameField();
```

.NET

```
String MetadataNameField { get; }
```

Get the name of the tupled PHRASE field to receive extracted meta-data names.

Property: TdbNlpInfo:MetadataValueField

Type: String
Access: Read

Java

```
String getMetadataValueField();
```

.NET

```
String MetadataValueField { get; }
```

Get the name of the tupled PHRASE field to receive extracted meta-data values.

Method: TdbNlpInfo:MapMetadataField

Type: void
Throws: TdbException

Java

```
void mapMetadataField(String categoryName  
                      String fieldName);
```

.Net

```
void MapMetadataField(String categoryName  
                      String fieldName);
```

Define a field to hold the values for a particular meta-data category.

Please refer to the NLP adapter's documentation for valid category names.

Method: TdbNlpInfo:AddInputField

Type: void
Throws: TdbException

Java

```
void addInputField(String fieldName);
```

.Net

```
void AddInputField(String fieldName);
```

Add the name of a TEXT or PHRASE field on which to base meta-data processing. The aggregated value of several fields can be used.

Text Analysis Example

The example below updates an existing record with keywords and metadata extracted from a text field in the record.

Java

```
// Retrieve existing record nr 123. Note that we do not
// require any field values to be fetched for this procedure.
TdbRecord record = new TdbRecord(session, "COFDEMO", false);
record.setRecordId(123);
record.get();

// Define NLP analysis behavior
TdbNlpInfo nlp = record.getNlpInfo();
nlp.setKeywordField("KEYWORDS");
nlp.setExtractKeywords(true);
nlp.setFieldMappingType(TdbNlpInfo.FieldMappingType.Tuples);
nlp.setMetadataTupleFields("META_NAME", "META_VALUE");
nlp.setExtractMetadata(true);
nlp.addInputField("FILE_TEXT");

// Analysis performed on server as part of commit operation.
record.commit();
```

20. Using JSON/XML Databases

Overview

TRIP contains a hybrid JSON and XML storage and search solution with the following features:

- Ability to represent and store XML and JSON documents with their complete structure.
- One single TRIP database design supports all kinds of XML and JSON documents, plus any kind of unstructured document or file.
- Support TRIP queries to find XML and JSON documents and sections within the documents.
- TRIP queries against a JSON/XML database can utilize the structure of the stored data.
- Optional storage of related DTD or schema for validation and stylesheet files for rendering of XML documents
- The contents and attributes of documents that are neither XML nor JSON are automatically extracted and indexed if TRIPcof is installed on the server.
- For hits in text nodes, hit markup can be applied so that the XML documents can be rendered with highlighting by the application (e.g. using XSL style sheets).

The XML-related functionality was prior to TRIP 8 available in a separately installed add-on product called TRIPxml. This is now integrated in TRIPsystem. Installing TRIPxml with TRIPsystem 8.0 or later is not needed nor supported.

API overview

Enumeration `TdbXmlRecord.IOMode`

The `IOMode` enumeration is used to control how document data is sent between the client and the server.

Enumeration: `TdbXmlRecord.IOMode`

Located in: `data`

The values within this enumeration are as follows:

`IOMode:Inline`

Transmission of document data will be inline (base64-encoded) in XPI requests and responses. This is the default.

`IOMode:File`

Transmission of document data will be to and from a file. This mode is only valid if the client and the server are running on the same machine.

IOMode:Stream

Transmission of document data will be streamed via a temporary HTTP connection between the client and the server.

Enumeration TdbXmlRecord.XmlRecordType

The XmlRecordType enumeration is used to indicate the file format of the document to import to a TRIPxml database.

Enumeration: TdbXmlRecord.XmlRecordType

Located in: data

The values within this enumeration are as follows:

XmlRecordType:UNKNOWN

The format of the file is not known. The client will assume that any file having a suffix ".xml" or ".xsl" is an XML file, and any other file is not. This is the default.

XmlRecordType:XML

The document is an XML file.

XmlRecordType:JSON

The document is a JSON file.

XmlRecordType:NONXML

The document is not in XML format. A DTD file is an example of a non-XML document usually stored in a TRIPxml database (for validation purposes).

Class TdbXmlRecord

The TdbXmlRecord class is a subclass of TdbRecord and represents a record in a JSON/XML database.

Class: TdbXmlRecord

Derived from: TdbRecord
Located in: data

The TdbXmlRecord class can be used just like the TdbRecord class, but its use is exclusive for JSON/XML databases.

Constructor: TdbXmlRecord*Java*

```
TdbXmlRecord(TdbSession session)

TdbXmlRecord(TdbSession session,
              TdbDatabaseDesign design,
              boolean createFields)

TdbXmlRecord(TdbSession session,
              String name,
              boolean createFields)
```

.Net

```
TdbXmlRecord(TdbSession session)

TdbXmlRecord(TdbSession session,
              TdbDatabaseDesign design,
              bool createFields)

TdbXmlRecord(TdbSession session,
              String name,
              bool createFields)
```

Construct a TdbXmlRecord to use for JSON and XML import and export operations. The simple version is sufficient for JSON/XML databases without custom fields. For such databases with custom fields, one of the two more elaborate versions are recommended. Please note that the database specified by name or design object cannot be anything else than a JSON/XML database.

Property: TdbXmlRecord:FileName

Type: String
Access: Read

Java

```
String getFileName()
```

.NET

```
String FileName { get; }
```

Contains the name of the exported file.

After a the contents of a JSON/XML record has been exported to a file, this property contains the name of the file. The intended use is in a situation when a directory name has been specified as export location, but no file name is available. A file name will be generated at that point, and the generated file name is provided via this property.

Property: TdbXmlRecord:HitMarkup

Type: Boolean
Access: Read, Write

Java

```
boolean getHitMarkup()  
void setHitMarkup(boolean hitMarkup)
```

.NET

```
bool HitMarkup { get; set; }
```

Indicates if the exported XML document is to have hit markup applied.

This property is used when the TdbXmlRecord instance is applied as a retrieval template for a TdbRecordSet. Hit markup consists of a pair of XML elements TRIPHLBEGIN and TRIPHLEND surrounding hits in XML text nodes. Highlighting can be produced by the application using hit markup, e.g. via an XSL stylesheet.

Property: TdbXmlRecord:InputOutputMode

Type: TdbXmlRecord.IOMode
Access: Read, Write

Java

```
TdbXmlRecord.IOMode getInputOutputMode()  
void setInputOutputMode(TdbXmlRecord.IOMode mode)
```

.NET

```
TdbXmlRecord.IOMode InputOutputMode { get; set; }
```

The IOMode enumeration is used to control how document data is sent between the client and the server. The default and recommended value for this property is IOMode.Inline, which causes document content to be sent as Base64-encoded strings in the XPI requests and responses.

Property: TdbXmlRecord:NetworkStreamPort

Type: int
Access: Read, Write

Java

```
int getNetworkStreamPort()  
void setNetworkStreamPort(int port)
```

.NET

```
int NetworkStreamport { get; set; }
```

The number of the TCP port to use for network streaming based import and export operations. A value of 0 (zero) will cause a port number to be dynamically allocated.

If there is a firewall between the client and the server, make sure that you specify a port number from a range that is enabled in the firewall. Remember to set the InputOutputMode property to Stream in order to use network streaming.

Also remember that in multithreaded situations, there may be multiple TdbXmlRecord exports and imports active simultaneously. Unless the port number is set to 0 (zero), the application is in such cases responsible for making sure that only available port numbers are used.

Property: TdbXmlRecord:RecordType

Type: TdbXmlRecord.XmlRecordType
Access: Read

Java

```
TdbXmlRecord.XmlRecordType getRecordType()
```

.NET

```
TdbXmlRecord.XmlRecordType RecordType { get; }
```

After a successful export, this property indicates the type of document that the record contains. Prior to import after a successful call to the PrepareImport method, this property indicates the type of document to be imported.

Property: TdbXmlRecord:StoreCopy

Type: Boolean
Access: Read, Write

Java

```
boolean getStoreCopy()  
void setStoreCopy(boolean storeCopy)
```

.NET

```
bool StoreCopy { get; set; }
```

Determines if a binary copy of an imported XML document is to be stored, or only the parsed version of it. This property is ignored for JSON documents.

Property: TdbXmlRecord:Stream

Type: Stream
Access: Read

Java

N/A

.NET

```
Stream Stream { get; }
```

Returns the stream that receives data from the XML record.

This is usually the stream as specified as argument to PrepareExport(Stream). However, in certain circumstances (e.g. when no file and no stream has been specified by the application) this property may return a MemoryStream instance with the received data.

Property: TdbXmlRecord:OutputStream

Type: OutputStream
Access: Read

Java

```
OutputStream getOutputStream()
```

.NET

N/A

Returns the stream that receives data from the XML record.

This is usually the stream as specified as argument to PrepareExport(OutputStream). However, in certain circumstances (e.g. when no file and no stream has been specified by the application) this property may return a ByteArrayOutputStream instance with the received data.

Property: TdbXmlRecord:UrlAlias

Type: String
Access: Read, write

Java

```
String getUrlAlias()
void setUrlAlias(String urlAlias)
```

.NET

```
String UrlAlias { get; set; }
```

The URL alias for the document.

The URL alias is the "real" URL of the document, i.e. the URL that is used to link two documents together via an XLink, for example. The URL alias is identical to the URL base and filename unless the URL alias has been explicitly specified to be something else during the import process.

Property: TdbXmlRecord:UrlBase

Type: String
Access: Read

Java

```
String getUrlBase()
```

.NET

```
String UrlBase { get; }
```

This method returns the URL base of the document, i.e. the URL without the actual file name.

If the document was imported directly from a file (e.g. using the server-side txput tool, or by using IOMode.File as InputOutputMode), then this property will be empty.

Property: TdbXmlRecord:Validate

Type: Boolean
Access: Read, Write

Java

```
boolean getValidate()  
void setValidate(boolean validate)
```

.NET

```
bool validate { get; set; }
```

Determines if XML documents are to be validated upon import.

Method: TdbXmlRecord:Clear

Type: void
Throws: N/A

Java

```
void clear(boolean all)
```

.Net

```
void Clear(bool all)
```

Clear any existing state from the TdbXmlRecord instance.

If the parameter 'all' is set to true, all state is cleared, including the database with which the record is associated, the record name/ID, etc. If false, the record structure is initialized, but the basic state of the record's association is maintained.

This method must always be called by the application when it is done with the record object if the I/O mode `IOMode.Stream` has been used. This is especially important if something has gone wrong during import or export. Failure to call this method may result in hanging threads.

Method: TdbXmlRecord:PrepareExport

Type: void
Throws: TdbException

Java

```
void prepareExport()  
void prepareExport(OutputStream stream)  
void prepareExport(String fileOrDir)
```

.Net

```
void PrepareExport()  
void PrepareExport(Stream stream)  
void PrepareExport(String fileOrDir)
```

Prepares to export the contents of the record. The actual export is done when the `get()` method is called.

The first and simplest overloaded version of the method exports the contents of the record to a `MemoryStream` (.NET) or `ByteArrayOutputStream` (Java), accessible via the `Stream` and `OutputStream` properties, respectively. This is a special case of the overloaded version of the method that explicitly takes a stream as argument. Both mentioned overloads require use of either `IOMode.Inline` or `IOMode.Stream` as value for the `InputOutputMode` parameter.

The third overloaded version of the method takes a string that can refer to a file or a directory. If the parameter is a directory, the name of the file (minus the path) will be read from the `D_DOCNAME` field and created in the specified directory. In any other case, the parameter is assumed to refer to a file to be created during the export procedure.

Method: `TdbXmlRecord.PrepareImport`

Type: `void`
Throws: `TdbException`

Java

```
void prepareImport(OutputStream stream,  
                  TdbXmlRecord.XmlRecordType type,  
                  String fileName)
```

.Net

```
void PrepareImport(Stream stream,  
                  TdbXmlRecord.XmlRecordType type,  
                  String fileName)
```

Prepares to import a document into a TRIPxml record. The actual import is done when the `commit()` method is called.

If the type parameter is `XmlRecordType.UNKNOWN`, then this method will try to ascertain the type from the provided file name. If the file name is not specified and the type is `XmlRecordType.UNKNOWN`, this method will throw an exception.

Direct file import (`InputOutputMode` set to `IOMode.File`) is possible if the stream parameter is set to null, the filename refers to an existing local file and the client application is running on the same machine as the TRIP server the current session is connected to.

Always have the application specify the name of the file if it is known. This also applies if the stream parameter is not null.

Insert XML documents

Inline import

The inline I/O mode (`IOMode.Inline`) is the default. For import operations this means that the document to be imported is converted to Base64 and inserted into the XPI request. The server will decode, parse and store the document.

Applications should always use this mode unless one of the special scenarios for direct file import or stream import applies.

While being useful in most scenarios, this mode is especially recommended for the import of moderately sized documents that do not use external entities and (in case of validation

during import) refer to DTDs or XML schemas that already have been imported into the same JSON/XML database.

Example:

```
C#

using TietoEnator.Trip.Nxp.Database;
using TietoEnator.Trip.Nxp.Data;

public void InlineImport(TdbSession session,
                        TdbDatabaseDesign db,
                        String xmlFileName)
{
    // Open a stream to the file to be imported.
    Stream inputStream = new System.IO.FileStream(strFile,
                                                FileMode.Open,
                                                FileAccess.Read,
                                                FileShare.Readwrite);

    // Create a new TdbXmlRecord object
    TdbXmlRecord rec = new TdbXmlRecord(session,db,false);

    // Specify that we'll use Inline mode to import.
    rec.InputOutputMode = TdbXmlRecord.IOMode.Inline;

    rec.StoreCopy = true; // Store a copy in the database
    rec.Validate = false; // Don't validate the document

    // Prepare the TdbXmlRecord instance to perform an XML
    // import upon the next call to Commit.
    rec.PrepareImport ( inputStream,
                        TdbXmlRecord.XmlRecordType.XML,
                        xmlFileName);

    // Performs the actual import
    rec.Commit();
}
```

Direct file import

The scenario for direct file import (the I/O mode `IOMode.File`) is usable when the client and the server are located on the same machine. This is the case if either:

- An instance of the `TdbLocalSession` class is used as the session object.
- The address of the host connected to via `TdbTripNetSession` is the loopback address (127.0.0.1) or one of the addresses associated with one of the network interfaces of the local machine.

If either of the two conditions above is met, then feel free to use the `IOMode.File` as the value for the `InputOutputMode` property.

Note that if the `TdbTripNetSession` is used to connect to a TRIP installation on the local host, make sure that the user that the `tbserver` process is set up to run as has read access to the file to be imported.

This mode is especially recommended for:

- the import of very large documents, as no network read or write buffers are required or used for the file data when this mode is active
- the import of documents that refer to other files, such as entity files and unimported DTDs and XML schemas

Example:

```
C#  
  
using TietoEnator.Trip.Nxp.Database;  
using TietoEnator.Trip.Nxp.Data;  
  
public void DirectFileImport(TdbSession session,  
                             TdbDatabaseDesign db,  
                             String xmlFileName)  
{  
    // Create a new TdbXmlRecord object  
    TdbXmlRecord rec = new TdbXmlRecord(session,db,false);  
  
    // Specify that we'll use File mode to import.  
    rec.InputOutputMode = TdbXmlRecord.IOMode.File;  
  
    rec.StoreCopy = true; // Store a copy in the database  
  
    rec.Validate = false; // Don't validate the document  
  
    // Prepare the TdbXmlRecord instance to perform an XML  
    // import upon the next call to Commit.  
    rec.PrepareImport ( null,  
                        TdbXmlRecord.XmlRecordType.XML,  
                        xmlFileName );  
  
    // Performs the actual import  
    rec.Commit();  
}
```

Stream import

Stream import (the I/O mode `IOMode.Stream`) is usable for the occasional import of very large documents when the client and the server are located on separate machines.

During stream import, the client creates a dedicated HTTP listener on a TCP socket whose port number is dynamically allocated or explicitly specified by the application. The XPI request sent to the server indicates the temporary URL from which the document is to be read. If the document refers to other files (e.g. external entities), these files are also requested. When TRIP is finished with the import, the listener shuts down.

Please note that if an exception occurs while importing in stream mode, the HTTP listener, which is running on its own thread, may not be properly shut down. In order to avoid hanging HTTP listener threads, always be sure to call the Close method on the TdbXmlRecord object used for the import operation when the import has finished (successfully or otherwise).

Example:

```
C#
using TietoEnator.Trip.Nxp.Database;
using TietoEnator.Trip.Nxp.Data;

public void StreamImport(TdbSession session,
                        TdbDatabaseDesign db,
                        String xmlFileName)
{
    // Create a new TdbXmlRecord object
    TdbXmlRecord rec = new TdbXmlRecord(session,db,false);

    rec.NetworkStreamPort = 0; // Use dynamic port

    // Specify that we'll use stream mode to import.
    rec.InputOutputMode = TdbXmlRecord.IOMode.Stream;

    rec.StoreCopy = false; // No copy in database

    rec.validate = false; // Don't validate the document

    // Prepare the TdbXmlRecord instance to perform a TRIPxml
    // import upon the next call to Commit.
    rec.PrepareImport ( null,
                       TdbXmlRecord.XmlRecordType.XML,
                       xmlFileName );

    // Performs the actual import
    rec.Commit();
}
```

Insert unstructured files in JSON/XML databases

JSON/XML databases can also store unstructured files. If TRIPcof or TRIPview-C is also installed on the server, the document text and properties are automatically extracted and stored in the database.

If DTD files are stored, they will be handled as unstructured files, but with the added benefit that they can be used by during validation of any XML document that is written according to the DTD.

The following fields are exclusive to the storage of non-XML data:

- DAV_NONXMLBLOB A binary copy of the unstructured file.
- D_DOCTEXT Extracted text of the file.
- D_PROPNAME Extracted property names.
- D_PROPVALUE Extracted property values.

Inline import

Inline import of an unstructured file means that the file is converted to Base64 and inserted into the XPI request. The server will decode the document and store it, optionally using TRIPcof or TRIPview-C to extract text and document properties.

This mode is default. It is recommended that applications always use this mode unless one of the special scenarios for direct file import applies.

The use of the TdbXmlRecord class is identical to when inline import of XML documents is performed, except for the second parameter to the PrepareImport method, which must be set to NONXML

Example:

C#

```
// See XML inline import example for the rest of the code.  
rec.PrepareImport ( inputStream,  
                  TdbXmlRecord.XmlRecordType.NONXML,  
                  fileName );
```

Direct file import

The scenario for direct file import (the I/O mode IOMode.File) is usable for non-XML files when the client and the server are located on the same machine. This is the case if either:

- An instance of the TdbLocalSession class is used as the session object.
- The address of the host connected to via TdbTripNetSession is the loopback address (127.0.0.1) or one of the addresses associated with one of the network interfaces of the local machine.

If either of the two conditions above is met, then feel free to use the IOMode.File as the value for the InputOutputMode property.

Note that if the TdbTripNetSession is used to connect to a TRIP installation on the local host, make sure that the user that the tbserver process is set up to run as has read access to the file to be imported.

The use of the TdbXmlRecord class is identical to when inline import of XML documents is performed, except for the second parameter to the PrepareImport method, which must be set to NONXML

Example:

C#

```
// See XML direct file import example for the rest of  
// the code.  
rec.PrepareImport ( null,  
                  TdbXmlRecord.XmlRecordType.NONXML,  
                  fileName );
```

Retrieve documents from a JSON/XML database

The procedure for retrieving documents from a JSON/XML database is similar to that of retrieving any other type of record from TRIP. Either use `TdbSearch` or `TdbRecordSet` to search for the records to retrieve, or use the `TdbXmlRecord` directly to fetch a record based on its record ID.

When exporting a structured document, i.e. an XML or JSON document, the `RecordType` property should be assigned before retrieval to specify the format in which the document should be returned. If left unspecified, XML will be used.

Inline export

Inline export means that the file to be exported is converted to Base64 and inserted into the XPI response. The client will decode the document and pass it on the application.

This mode is default. It is recommended that applications always use this mode unless one of the special scenarios for direct file export or stream export applies.

Example:

```
C#

using System.IO;
using TietoEnator.Trip.Nxp.Database;
using TietoEnator.Trip.Nxp.Data;

public void InlineExport(TdbSession session,
                        TdbDatabaseDesign db,
                        int recordId,
                        String outputFileName)
{
    // Create a new TdbXmlRecord object
    TdbXmlRecord rec = new TdbXmlRecord(session,db,false);

    rec.RecordId = recordId; // The ID of the record to export

    // Use inline mode for the export
    rec.InputOutputMode = TdbXmlRecord.IOMode.Inline;

    // Specify the file format to export the record as
    rec.RecordType = TdbXmlRecord.TdbXmlRecordType.XML;

    // Open an output stream to receive the data.
    FileStream outStream = new FileStream(m_ outputFileName,
        FileMode.Create, FileAccess.Write,
        FileShare.ReadWrite);

    // Prepare the TdbXmlRecord instance to perform a TRIPxml
    // export upon the next call to Get.
    rec.PrepareExport(outStream);

    // Perform the export.
    rec.Get();
}
```

Direct file export

The scenario for direct file export (the I/O mode `IOMode.File`) is usable when the client and the server are located on the same machine. This is the case if either:

- An instance of the `TdbLocalSession` class is used as the session object.
- The address of the host connected to via `TdbTripNetSession` is the loopback address (127.0.0.1) or one of the addresses associated with one of the network interfaces of the local machine.

If either of the two conditions above is met, then feel free to use the `IOMode.File` as the value for the `InputOutputMode` property.

Note that if the `TdbTripNetSession` is used to connect to a TRIP installation on the local host, make sure that the user that the `tbserver` process is set up to run as has write access to the file to be imported.

This mode is especially recommended to export very large documents, as no network read or write buffers are required or used for the file data when this mode is active.

Example:

```
C#
using TietoEnator.Trip.Nxp.Database;
using TietoEnator.Trip.Nxp.Data;

public void DirectFileExport(TdbSession session,
                             TdbDatabaseDesign db,
                             int recordId,
                             String outputFileName)
{
    // Create a new TdbXmlRecord object
    TdbXmlRecord rec = new TdbXmlRecord(session, db, false);

    rec.RecordId = recordId; // The ID of the record to export

    // Use inline mode for the export
    rec.InputOutputMode = TdbXmlRecord.IOMode.File;

    // Specify the file format to export the record as
    rec.RecordType = TdbXmlRecord.TdbXmlRecordType.XML;

    // Prepare the TdbXmlRecord instance to perform an
    // export upon the next call to Get.
    rec.PrepareExport(outputFileName);

    // Perform the export. After successful completion of this
    // command, the requested document has been exported to the
    // specified file.
    rec.Get();
}
```

Stream export

Stream export (the I/O mode `IOMode.Stream`) is usable for the occasional export of very large documents when the client and the server are located on separate machines.

During stream export, the client creates a dedicated HTTP listener on a TCP socket whose port number is dynamically allocated or explicitly specified by the application. The XPI request sent to the server indicates the temporary URL to which the document is to be posted. When the export is complete, the listener shuts down.

Please note that if an exception occurs while exporting in stream mode, the HTTP listener, which is running on its own thread, may not be properly shut down. In order to avoid hanging HTTP listener threads, always be sure to call the Close method on the TdbXmlRecord object used for the export operation when the export has finished (successfully or otherwise).

Example:

```
C#

using System.IO;
using TietoEnator.Trip.Nxp.Database;
using TietoEnator.Trip.Nxp.Data;

public void InlineExport(TdbSession session,
                        TdbDatabaseDesign db,
                        int recordId,
                        String outputFileName)
{
    // Create a new TdbXmlRecord object
    TdbXmlRecord rec = new TdbXmlRecord(session,db,false);

    rec.RecordId = recordId; // The ID of the record to export
    rec.NetworkStreamPort = 0; // Use dynamic port

    // Use inline mode for the export
    rec.InputOutputMode = TdbXmlRecord.IOMode.Stream;

    // Specify the file format to export the record as
    rec.RecordType = TdbXmlRecord.TdbXmlRecordType.XML;

    // Prepare the TdbXmlRecord instance to perform an
    // export upon the next call to Get.
    rec.PrepareExport(outputFileName);

    // Perform the export.
    rec.Get();
}
```

Update JSON/XML database records

The procedure for updating JSON/XML database records is the same as for creating new ones; to import the required file. You need to make sure that the record instance is referring to an existing record, however. To do this assign the RecordId property to the ID of the record you wish to update, or use a record instance as returned from a search using the TdbRecordSet class.

For a detailed code example, please refer to the TRIPnXP sample program XmlPut or the TRIPjXP example class XmlExport.

Using XPath Queries

While CCL can be used to query JSON/XML databases, using XPath provides a much better control of the query against such data. XPath queries work over all structured records (JSON and XML) in the currently open database. For a description of the syntax, refer to the document "JSON and XML Databases", available in the TRIPsystem documentation set.

Using XPath with the TdbSearch Class

The ExecuteXPath method was added to the TdbSearch class in version 2.1-0. This method takes an XPath expression that is evaluated to a TRIP search set matching the specified expression.

Method: TdbSearch:ExecuteXPath

Type: void
Throws: TdbException

Java

```
void executeXPath(String xpathStatement)
```

.Net

```
void ExecuteXPath(String xpathStatement)
```

This method executes an XPath statement as a query. A JSON/XML database must be open in order for this to work.

Use ExecuteXPath method just like you would use the Execute method. The behavior with respect to TdbSearch state and usage is the same. However, for retrieval of records, you should either export the JSON or XML documents (see page 216), or retrieve the search results as an XML fragment set (see page 221).

C#

```
using TietoEnator.Trip.Nxp.Data;

void XPathSearchTest (TdbSearch searchHandler)
{
    searchHandler.ExecuteXPath("//para[@lang='EN']");
    TdbSearchSet searchSet = searchHandler.LastSearchSet;
    Console.WriteLine("{1} record(s)", searchSet.RecordCount);
}
```

Using XPath with the TdbRecordSet Class

In version 2.1 there is a new property IsQueryXPath (Java) and QueryIsXPath (.NET) that you can use to specify that the query statement assigned to the Query property is an XPath expression and should be executed as such.

Property: TdbRecordSet:QueryIsXPath

Type: boolean
Access: Read, Write

Java

```
boolean isQueryXPath()
void setQueryXPath(boolean enable);
```

.Net

```
bool QueryIsXPath { get; set; }
```

Establish whether the query statement is an XPath expression.

So to execute an XPath query via the TdbRecordSet class, the only thing that sets the procedure apart from executing CCL queries is that you need to set the aforementioned property to true before calling get().

Java

```
TdbRecordSet rs = new TdbRecordSet(session);
rs.setDatabase("myxmlbase");
rs.setQuery("//para[@lang='En']");
rs.setQueryXPath(true);
rs.setFrom(1);
rs.setTo(5);
rs.get();
```

VB.Net

```
Dim rs as New TdbRecordSet(session)
rs.Database = "myxmlbase"
rs.Query = "//para[@lang='En']"
rs.QueryIsXPath = true
rs.From = 1
rs.To = 5
rs.Get()
```

Using XPath with the TdbCclCommand Class

The ExecXPath method was added to the TdbCclCommand class in version 2.1-0. This method takes an XPath expression that TRIP evaluates to a search set matching the specified expression.

Method: TdbCclCommand:ExecXPath

Type: void
Throws: TdbException

Java

```
void execXPath(String xpathStatement)
```

.Net

```
void ExecXPath(String xpathStatement)
```

Executes an XPath statement as a query. A JSON/XML database must be open in order for this to work.

Retrieving Search Results as XML Fragment Sets

If you have a search set that contains XML and/or JSON documents, and you only want to get fraction of the contents from each document, you can ask TRIP to generate such an extract itself instead of you having to retrieve all documents to your application and process them there. What you get is a single XML document that contains fragments from the documents in the search set.

Method: TdbSearchSet:AsFragmentSet

Type (Java): XmlDocument
Type (.NET): org.w3c.dom.Document
Throws: TdbException

Java

```
Document asFragmentSet(String xpathExpression,  
                        int from,  
                        int to,  
                        boolean hitMarkup);
```

.Net

```
XmlDocument AsFragmentSet(string xpathExpression,  
                           Int32 from,  
                           Int32 to,  
                           bool hitMarkup);
```

This method retrieves a document containing XML fragments from the specified range of records in the current search set, optionally with hit markup applied.

The retrieved fragments will only be those matched by the supplied XPATH expression; if this expression does not match a certain record in the current search set, no fragments (and hence no information) from that record will be retrieved.

Example of fragment set retrieval:

```
C#
using TietoEnator.Trip.Nxp.Data;

XmlDocument FragmentTest (TdbSearchSet sset)
{
    return sset.AsFragmentSet("//para[1]",1,2,false);
}
```

The XML document produced by the above example could look like this:

XML Fragment Set Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<tripxml:FRAGMENTSET
  xmlns:tripxml="http://www.tieto.com/trip/xml"
  tripxml:FRAGMENTCOUNT="2" tripxml:RECORDCOUNT="129"
  tripxml:SEARCHSET="1">
  <tripxml:FRAGMENT tripxml:DATABASE="XPATHTEST" tripxml:RID="1">
    <para lang="EN">I love money, but not that despicable
    Mr. Simpson.</para>
  </tripxml:FRAGMENT>
  <tripxml:FRAGMENT tripxml:DATABASE="XPATHTEST" tripxml:RID="2">
    <para>welcome to TRIPxml version 3.0-0!</para>
  </tripxml:FRAGMENT>
</tripxml:FRAGMENTSET>
```

Although the above example shows one fragment from each document, any number of fragments may actually be returned from a single document (even none at all). The number of fragments produced from each document depends entirely on the XPath expression used to select the fragments. The fragments from a single document always come together and in document order.

Note that the attribute FRAGMENTCOUNT is not the number of fragments, but the number of *records* matched by the XPath expression used to extract the fragments.

21. Facets

Definition

Attributes or other characteristics of objects that when used to classify objects often are referred to as facets. Such classification is used in faceted search to enable the user to easily navigate and drill down into a data set.

Facets in TRIP

A facet is a classification by the above definition. The values of a facet are in the case of TRIP taken from the live data in the currently opened database or cluster. There is no separate facet definition that needs to be set up. Facets can be set up, used and dropped without any sort of design or other definition required in advance.

Facet Classes

There are four facet classes in TRIPnxp and TRIPjxp, all of which derive from the abstract base class TdbFacet:

Class: TdbFacet

Derived from: IEnumerable
Located in: facet

A database for which a TRIP classification scheme been enabled can be used with the TdbClassificationFacet. The values for this facet class are the automatically assigned classification categories.

Class: TdbClassificationFacet

Derived from: TdbFacet
Located in: facet

The values of fields in TRIP databases can be used as facet values using the TdbFieldTermFacet class. It generates facets in a fashion quite similar to the way term lists are generated by a display order.

Class: TdbFieldTermFacet

Derived from: TdbFacet
Located in: facet

The TdbKvpFacet class generates facets based on a key/value pair (KVP) display order, using the values from one field as facet names, tupled with another field that contains the facet values.

Class: TdbKvpFacet

Derived from: TdbFacet
Located in: facet

Some facets are of a type that have no explicit values that can be used directly, but must be derived instead. TRIPnxp and TRIPjxp offers one way to do this using the TdbSearchFacet class. Each facet value is one search result. For example, date ranges can be used as facets using this class.

Class: TdbSearchFacet

Derived from: TdbFacet
Located in: facet

Using TRIP Facets

The TdbFacetSet Class

A facet search user interface tend to use multiple facets. Because of this, the facet implementation in TRIPnxp and TRIPjxp contains the class TdbFacetSet.

Class: TdbFacetSet

Derived from: TdbMessageProvider
Located in: facet

This class is the means by which facet values are retrieved from TRIP. You prepare it by adding TdbFacet instances to it that define the facets to retrieve values for. Then values for all facets are retrieved at the same time. If a facet happens to be associated with a lot of values, only the first few values will actually be fetched immediately. The remaining values will be automatically retrieved as they are requested by the application.

The TdbFacetValue Class

The values of a facet are retrieved from the facet classes in the form of instances of TdbFacetValue.

Class: TdbFacetValue

Derived from: Object
Located in: facet

The easiest way to use it is to use the facet instance (i.e. TdbFacet subclass instance) as a collection and iterate over it.

Facet Example

Generate a field term facet with the 10 most frequent values from the PERSON field in the ALICE database, but with minimum frequency set to 2.

C#

```
TdbFacet facet = new TdbFieldTermFacet("PERSON");  
facet.Database = "ALICE";  
facet.SortOrder = TdbFacetSortOrder.ValueAscending;  
facet.SetFrequencyLimits(10, 0);  
  
TdbFacetSet fs = new TdbFacetSet(session);  
fs.Add(facet);  
fs.Get();
```

It is also possible to take the facet values from a search set instead of directly from a database. To do this, replace the "facet.Database" assignment with an assignment of the "facet.SearchSet" property.

NOTE: The *FetchFrom* and *FetchTo* properties have been **DEPRECATED** without replacement. These values are no longer used, and the associated properties will be removed in a future release. All retrieval of facet value ranges is now fully automatic. If your application sets *FetchFrom* and/or *FetchTo*, it must be modified. Use the *GetValue* method on the *TdbFacet* instance to access the values you need.

Facet Baselines

Version 7.2-1 of TRIPsystem and TRIPjxp/TRIPnxp introduced a feature referred to as “facet baseline”. This makes it possible to recall a previous, initial set of facet values so that a new facet request can 1) include the previous and 2) add values from the baseline to the new result if the new result is smaller than the requested range.

Each facet in a TdbFacetSet is individually configured for baseline use. The first step is to define a baseline. This is done when then facets are configured for retrieval as a facet set, using the new TdbFacet properties RegisterBaseline, BaselineSize and BaselineKey.

Property: TdbFacet:RegisterBaseline

Type: bool
Access: Read, write

Java

```
void setRegisterBaseline(boolean enable)

boolean isRegisterBaseline()
```

.Net

```
void RegisterBaseline { get; set; }
```

Register a new baseline by setting the property to true. A registration can be cancelled before the TdbFacetSet.Get method is called by setting this property back to false.

Property: TdbFacet:BaselineSize

Type: String
Access: Read, write

Java

```
int getBaselineSize()

void setBaselineSize(int size);
```

.Net

```
int BaselineSize { get; set; }
```

Set the number of facet values to store in the baseline for this facet. The default and minimum number is 100.

Property: TdbFacet:BaselineKey

Type: String
Access: Read, write

Java

```
String getBaselineKey()

void setBaselineKey(String key);
```

.Net

```
String BaselineKey { get; set; }
```

Set a key identifying the facet baseline to set or use. This is by default the same as the name supplied to the constructor. You will normally not have to set this

property, unless you have more than one field term facet for the same field, or you have a search facet against the same field as a field term facet.

When the facet set that includes the baseline definitions is executed, the facet results are stored as baselines. The baselines remain active and possible to use in the current session until either:

- A new baseline requested for the same facet. This will only affect the facet specifying the new baseline, although one would normally specify new baselines for all facets at the same time.
- The database open for search is changed. This affects all facets. After a BASE command (explicit or implicit), facet baselines should be re-registered.
- The application calls the `TdbFacetSet.removeBaselines` method.

The facet values returned as part of the request to establish a baseline can be used normally by the application. These facet value would typically represent an initial navigational state, e.g. after just having started a new session, or opening a new or different database or cluster.

Once the baseline is established, subsequent facet requests will have to specify that they wish to use the previously established baseline. This is done by using the new property `TdbFacet.UseBaseline` and (if needed) the `TdbFacet.BaselineKey` property.

Property: `TdbFacet:UseBaseline`

Type: `bool`
Access: Read, Write

Java

```
String isUseBaseline ()
void setUseBaseline(Boolean enable);
```

.Net

```
bool UseBaseline { get; set; }
```

Set to true to use a previously registered baseline for this facet.

A facet request with the `UseBaseline` property set to true must use the same definition as the facet used to establish the baseline. For classification, field and KVP facets this means the same field (the mask and search set may vary). For search facets, this means the order of the queries that comprise the facet.

NOTE: The *BaselineSize* property of the *TdbFacet* class has been **DEPRECATED** and is from version 8.0 of *TRIPsystem* no longer used. The assignment of this property is only valid for *TRIPsystem* 7.2.

Facet Baseline Registration Example

Register baselines for two facets.

```
C#  
  
TdbFacetSet fs = new TdbFacetSet(session);  
  
TdbFacet facet = new TdbFieldTermFacet("PERSON");  
facet.Database = "ALICE";  
facet.RegisterBaseline = true;  
fs.Add(facet);  
  
facet = new TdbFieldTermFacet("SPEAKER");  
facet.Database = "ALICE";  
facet.RegisterBaseline = true;  
fs.Add(facet);  
  
fs.Get();
```

As mentioned, this produces the same results as a normal facet request would. So after a successful call to `TdbFacetSet.get()`, the application would typically render the facets and facet values as it normally does.

Facet Baseline Usage Example

Request facet values that makes sure that values missing from the new request are included from the baseline, so that the number of values per facet remain fixed.

C#

```
// 'search' is here assumed to be instance of TdbSearch that
// previously has been used to open the database ALICE.
search.Execute("FIND CARPENTER");

TdbFacetSet fs = new TdbFacetSet(session);

TdbFacet facet = new TdbFieldTermFacet("PERSON");
facet.SearchSet = search.LastSearchSet.SearchId;
facet.UseBaseline = true;
facet.BaselineAtEnd = true;
fs.Add(facet);

facet = new TdbFieldTermFacet("SPEAKER");
facet.SearchSet = search.LastSearchSet.SearchId;
facet.UseBaseline = true;
facet.BaselineAtEnd = true;
fs.Add(facet);

fs.Get();
```

This example would without baselines result in 8 values for the PERSON field facet and 2 values for the SPEAKER field facet. But with baselines we get the complete list of 10 facet values returned for each facet (102 and 60, respectively).

Setting BaselineAtEnd to true means that we get the facet values that are present in the search first, and values that are only present in the baseline at the end of the list.

Facet Scrolling Usage Example

If the number of values for a facet is large, applications tend to only display a smaller range of values in the user interface. The values are typically rendered in a widget that allows scrolling. When the user scrolls, additional values are displayed.

The application does not necessarily have retrieved all values up front (and neither should it). If it doesn't have access to the same TdbFacet instances that originally were used to obtain the first values, the OpenExisting property can be used.

NOTE: This only works with baselined facets.

C#

```
TdbFacetSet fs = new TdbFacetSet(session);

TdbFacet facet = new TdbFieldTermFacet("PERSON");
facet.UseBaseline = true;
facet.OpenExisting = true;
facet.BaselineAtEnd = true;
fs.Add(facet);

facet = new TdbFieldTermFacet("SPEAKER");
facet.UseBaseline = true;
facet.OpenExisting = true;
facet.BaselineAtEnd = true;
fs.Add(facet);

fs.Get();
```

This example would recreate the TdbFacet instances for the latest facet request. The application can after this proceed to retrieve the values to be displayed in the user interface using the GetValue method. The benefit here is that this is very quick since no search or DISPLAY operation has to be performed to retrieve the facet values from the TRIP database – it is all there already.

22. Graphs

TRIP is from version 7.1 of TRIPsystem supporting graph databases. APIs for this are available in version 3.1 of TRIPjxp and TRIPnxp.

Graphs in TRIP

A graph is a structure that uses vertices, edges and properties to represent data. A graph database is a database that use graph structures.

When creating a database in TRIPmanager 7.1 and later, it is possible to choose to create a graph database. This kind of database design is primarily used by the graph implementation in TRIP in order to store and represent edges. An edge is a relation between vertices (nodes), and a vertex in the TRIP graph implementation is a regular record. Any kind of record in any database on the local system. The records used as vertices in a TRIP graph do not have to be explicitly created for use in a graph, nor do they have to be modified in order to be used.

Graphs in general can be directed or undirected. An edge in an undirected graph states that the associated vertices are related in a bidirectional manner. If A and B are related, it goes both ways. An edge in a directed graph is unidirectional - it only goes one way. In order to state bi-directionality in such a graph, two edges are required.

TRIP graphs are directional in order to enable the representation of graph data where the edge direction matters (e.g. a street map with some one-way streets).

Main Graph Classes

The main graph class in TRIPjxp and TRIPnxp is TdbGraph. An instance of this class represents a graph database and via this instance the application can access graph query and navigation operations.

Class: TdbGraph

Derived from: TdbMessageProvider
Located in: graph

The TdbGraph class does not provide all functionality itself. While it can be used to create graph edges and a simple kind of vertices, the main search and navigation functionality is available in the TdbGraphSet class, an instance of which can be obtained using the GraphSet property of the TdbGraph class.

Class: TdbGraphSet

Derived from: Object
Located in: graph

The TdbGraphSet class is abstract. Its subclasses are TdbGraphDatabase and TdbSubGraph.

Class: TdbGraphDatabase

Derived from: TdbGraphSet
Located in: graph

The TdbGraphDatabase class represents an entire graph database.

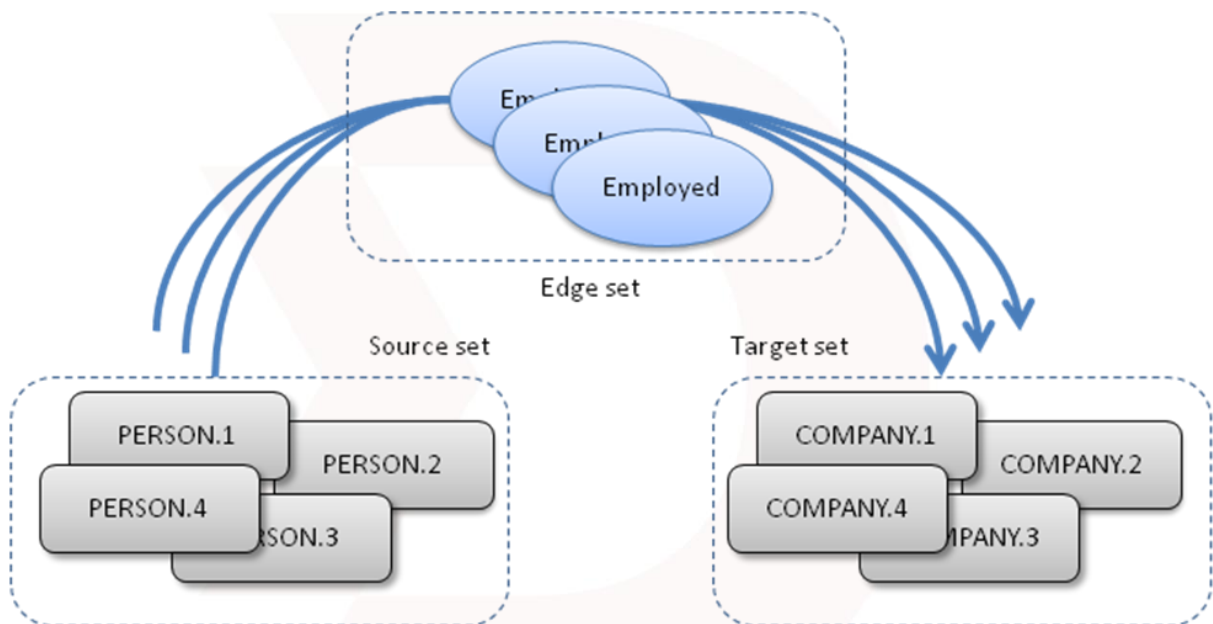
Class: TdbSubGraph

Derived from: TdbGraphSet
 Located in: graph

The TdbSubGraph class represents the result of a graph search operation, which in TRIP always is a subgraph.

Basic Graph Operations**Source, Target and Edge Sets**

TRIP is inherently set based, and so is its graph implementation. When executing a graph search or navigation operation, search sets of three types must be specified. The *source* set is a set of vertices that will act as starting point for the graph operation. The *edge* set is the set of graph edges to restrict your operations to. The *target* set is the set of vertices you want to restrict your results to.



The set number 0 (zero) indicates the "universal set", i.e. that no restrictions are to be imposed. In contrast to regular TRIP semantics, search set zero does not refer to the last executed search.

Note that if the edges are traversed backwards, the target and source sets switches roles because of the semantics of the operation.

Forward Navigation

Forward navigation starts from a set of one or more vertices and follows edges from there. Edges can be specified by name, e.g. FRIEND, LIKE, etc.

To use forward navigation, use the Follow method of the TdbGraphSet class.

Method: TdbGraphSet:Follow

Type: TdbGraphSet
Throws: TdbException

Java

```
TdbGraphSet follow(String edge, int targetSet);
```

.Net

```
TdbGraphSet Follow(string edge, int targetSet);
```

This method is called on the TdbGraphSet instance representing the edge set.

The Follow method works on the graph database itself. Normally, one or a few specific source vertices are required by the operation, so these must be selected before the Follow method can be invoked. This is done by calling the Source method. This turns a search set with the relevant vertices into a graph set where the vertex records are used as source vertices in the edge records in the graph.

Method: TdbGraphSet:Source

Type: TdbGraphSet
Throws: TdbException

Java

```
TdbGraphSet source(int searchSet);
```

.Net

```
TdbGraphSet Source(int searchSet);
```

This method is called on the TdbGraphSet instance representing the graph edge set.

An example. Assume that we have a set nr 3 that consists of regular TRIP records representing users in a social web site. We wish to find out what these users LIKE.

Java

```
TdbGraphSet edgesFromUsers = graph.source(3);  
TdbGraphSet likes = edgesFromUsers.follow("LIKE",0);
```

The set 'likes' now contains the LIKE edges from the users specified by set 3.

Backward Navigation

The TRIP graph representation uses directional, one-way edges. This means that it is possible to represent relationships between two objects that goes one way, but no both, in an unambiguous way.

This means that a bi-directional relationship in TRIP must have two edges. One in each direction. But sometimes that is not strictly necessary nor desired, but it still is so that some use cases involves following the relationships in the reverse direction.

Backward navigation thus involves following the edges in reverse direction. The method to use is called Backtrack:

Method: TdbGraphSet:Backtrack

Type: TdbGraphSet
Throws: TdbException

Java

```
TdbGraphSet backtrack(String edge, int sourceSet);
```

.Net

```
TdbGraphSet Backtrack(String edge, int sourceSet);
```

This method is called on the TdbGraphSet instance representing the graph edge set.

The Backtrack method works on the graph database itself. Normally, one or a few specific target vertices are required by the operation, so these must be selected before the Backtrack method can be invoked. This is done by calling the Target method. This turns a search set with the relevant vertices into a graph set where the vertex records are used as target vertices in the edge records in the graph.

Method: TdbGraphSet:Target

Type: TdbGraphSet
Throws: TdbException

Java

```
TdbGraphSet target(int searchSet);
```

.Net

```
TdbGraphSet Target(int searchSet);
```

This method is called on the TdbGraphSet instance representing the graph edge set.

An example. Assume that we have a set nr 3 that consists of regular TRIP records representing users in a social web site. We want to find out what other users LIKE the ones in set 3.

Java

```
TdbGraphSet edgesFromUsers = graph.target(3);  
TdbGraphSet likes = edgesFromUsers.backtrack("LIKE",0);
```

The set 'likes' now contains the LIKE edges to the users specified by set 3.

Resolving Vertices

The results of the Follow, Backtrack, Source and Target methods are all graph sets, i.e. they consist of edges. This is fine as long as the (navigation) operation continues within the graph. But eventually, the application will want to access a set with the actual vertex records. This is done using the methods ResolveTargets and ResolveSources.

Method: TdbGraphSet:ResolveTargets

Type: TdbSearchSet
Throws: TdbException

Java

```
TdbSearchSet resolveTargets();
```

.Net

```
TdbSearchSet ResolveTargets();
```

This method is called on the TdbGraphSet instance representing the graph edge set with the targets to resolve.

Method: TdbGraphSet:ResolveSources

Type: TdbSearchSet
Throws: TdbException

Java

```
TdbSearchSet resolveSources();
```

.Net

```
TdbSearchSet ResolveSources();
```

This method is called on the TdbGraphSet instance representing the graph edge set with the sources to resolve.

By adding target vertex resolution to the Forward navigation example, the code will look something like this:

Java

```
TdbGraphSet edgesFromUsers = graph.source(3);  
TdbGraphSet likes = edgesFromUsers.follow("LIKE",0);  
TdbSourceSet liked = likes.resolveTargets();
```

And the Backward navigation example with source vertex resolution will look like this:

Java

```
TdbGraphSet edgesFromUsers = graph.target(3);  
TdbGraphSet likes = edgesFromUsers.backtrack("LIKE",0);  
TdbSourceSet fans = likes.resolveSources();
```

In either case, the result is a TdbSourceSet object that refers to a regular TRIP search set with the records acting as sources and targets, respectively, in the resolved graph set.

Transitive Search

The previous examples used a social network type of scenario where users LIKE things. Such relationships are not transitive. If they were, it would mean that if user A likes B and user B likes C, then it would follow that user A likes C. This is clearly not automatically the case and is not normally the semantics of the LIKE relationship.

Some relationships are transitive. Take a road map that shows roads between cities, for example. The semantics of the ROAD relationship is clearly transitive. If somebody can drive between Düsseldorf and Cologne and between Cologne and Frankfurt, it follows that somebody can go from Düsseldorf to Frankfurt via Cologne.

Whether or not an edge (a relationship) is transitive is not something that TRIP enforces or controls. The semantics is up to the application to determine in this case.

While the Forward and Backtrack methods can be used in sequence to perform a transitive type of graph search, the Transitive method offers a quicker way to do this.

Method: TdbGraphSet:Transitive

Type: TdbSearchSet
Throws: TdbException

Java

```
TdbGraphSet transitive(int sourceSet, int targetSet,  
                      String[] edgeNames, int maxDepth,  
                      boolean reverse, boolean allEdges);
```

.Net

```
TdbGraphSet Transitive(int sourceSet, int targetSet,  
                      String[] edgeNames, int maxDepth,  
                      bool reverse, bool allEdges);
```

This method is called on the TdbGraphSet instance representing the sub to perform the transitive search within.

The *sourceSet* parameter indicates the regular search set that specifies the vertices from which to start the transitive search operation. This parameter is required and must not be zero.

The *targetSet* parameter can be used to restrict the transitive search to a specific set of vertices, e.g. destination(s) in a road map scenario. If no restriction is needed, pass zero to indicate the universal set.

The *edgeNames* parameters is an array of edge names. This specifies what edges to traverse. Pass null or an empty array to traverse all kinds of edges.

The *maxDepth* parameter can be used to limit the search depth. Pass zero to allow the entire graph to be traversed.

In order to perform a transitive search in reverse direction with regard to the edges, set the *reverse* parameter to true. Note that this swaps the roles of the *sourceSet* and *targetSet* parameters; *sourceSet* will indicate the starting points being the target vertices of the edges, and *targetSet* indicates the destination vertices being the source vertices to find.

Passing *true* to the *allEdges* parameter indicates that the returned graph set should contain all the edges traversed in the transitive search. Passing *false* will only return the final edges that refer to the destination vertices.

As an example of a transitive search we'll use a genealogy scenario. Assume that there is a graph with CHILD edges. A transitive search in a forward direction will find descendants, and in the reverse direction it will find ancestors. The following example shows how to find the children and grandchildren of a certain individual or individuals indicated by search set number 3.

Java

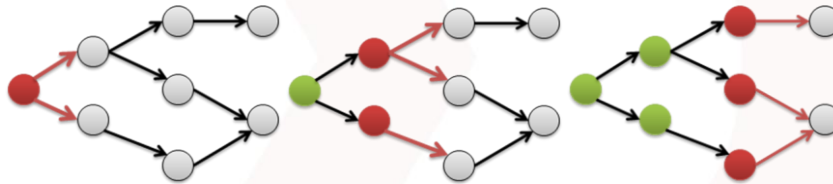
```
TdbGraphSet edges = graph.transitive(3,0,new String[]{"CHILD"},
                                     2,true,true);

TdbSourceSet descendants = edges.resolveSources();
```

Graph Path Analysis

Path analysis is a fairly common task when using graphs. The methods described previously in this chapter can be used for this purpose, but they leave it up to the application to string edges up to form paths, determine the shortest path, and avoid getting trapped in circular paths.

TRIP uses a set-based approach to path analysis. Instead of traversing each edge individually and analyze the vertex to find the outgoing edges for the next steps, TRIP processes all edges in parallel for all vertices at the same level of traversal depth. This is different than established algorithms such as Dijkstra's, but makes better use of TRIP's behavior than a more vertex oriented approach would.



The three small graphs in the above illustration shows how this type of path analysis works. All outgoing edges from the current source vertices at each stage are all analyzed simultaneously (current analysis step marked in red). Each such step generates one search set internally in TRIP.

TRIPjxp and TRIPnpx makes two path analysis methods available to the application; FindPaths, and ShortestPath. These methods return instances of the TdbGraphPath class.

Class: TdbGraphPath

Derived from: TdbSessionObject
Located in: graph

This class represents a path through a graph.

The FindPaths method:

Method: TdbGraphSet:FindPaths

Type: TdbGraphPath[]
Throws: TdbException

Java

```
TdbGraphSet findPaths(int startSet, String[] edgeNames,  
                      int targetSet, int maxDepth,  
                      boolean reverse);
```

.Net

```
TdbGraphSet FindPaths(int startSet, String[] edgeNames,  
                      int targetSet, int maxDepth,  
                      bool reverse);
```

This method calculates all paths associated with the current graph set that use vertices in the specified start set as origins (edge sources). This operation may take a long time to complete for large graphs. Consider raising the session timeout if the operation takes close to or more than one minute.

The ShortestPath method:

Method: TdbGraphSet:ShortestPath

Type: TdbGraphPath
Throws: TdbException

Java

```
TdbGraphSet shortestPath(int startSet,  
                         String[] edgeNames,  
                         int targetSet);
```

.Net

```
TdbGraphSet ShortestPath(int startSet,  
                         String[] edgeNames,  
                         int targetSet);
```

This method calculates the shortest path associated with the current graph. Paths are calculated between vertices in the specified start and target sets. It is possible to restrict the kind of edges to traverse by specifying an array of edge names. If the *edgeNames* parameters is not null and not an empty array, any edge whose name is not included will be ignored.

This analysis uses TRIP's own set-based edge approach and not a vertex based algorithm such as Dijkstra's.

Example of path analysis call:

Java

```
// Create search sets with start and destination
TdbSearch search = new TdbSearch(session);
search.execute("BASE CITYINFO");
search.execute("FIND CITY=Frankfurt");
int sourceSet = search.getLastSearchSet().getSearchId();
search.execute("FIND CITY=Cologne");
int targetSet = search.getLastSearchSet().getSearchId();

// Open the graph
TdbGraph graph = new TdbGraph(m_session, "ROADMAP");
TdbGraphSet gset = graph.getGraphSet();

// Perform the path analysis
TdbGraphPath route;
String[] edgeNames = new String[]{"ROAD", "HIGHWAY", "STREET"};
route = gset.shortestPath(sourceSet, edgeNames, targetSet);
```

Dealing with Lengthy Path Analysis Operations

A graph path analysis operation may take a very long time to complete if the graph is large or heavily interconnected. This section describes how to deal with such situations in the application.

Session Timeout

The default session timeout is set to one minute (60000 milliseconds). This is most likely to short for path analysis in even moderately sized graphs. To turn this up, use the timeout parameter to the TdbTripNetSession constructor.

For example, setting the timeout to 45 minutes:

Java

```
// Login to TRIP using a 45 minute read timeout
TdbSession session;

session = new TdbTripNetSession("localhost",23457,"",2700000);

session.login("me","secret");
```

Notificaitons

A notification is a type of signal sent from the server during the course of a lengthy operation. Graph analysis notifications are sent for each level deeper into the graph the analysis procedure steps and reports information on the current working state and the candidate results so far.

In order to set up the application to receive graph notifications, the abstract class `TdbNotificationSink` must be extended and the `onGraphComforter` method overridden. An instance of the subclass must then be passed to the current `TdbSession` object via the `NotificationSink` property. Finally, graph notifications are enabled by calling the `TdbSession` method `EnableNotification`.

Example of a simple `TdbNotificationSink` subclass:

Java

```
class GraphReport extends TdbNotificationsSink
{
    public GraphReport (TdbSession session) throws TdbException
    { super(session); }

    @Override
    public boolean onGraphComforter
        ( TdbGraphNotification notificationDetail )
    {
        // TODO: Use the notification detail object

        // Return true to continue with analysis and false to abort
        return true;
    }
}
```

Example of how to enable graph notifications:

Java

```
session.setNotificationSink(new GraphReport(session));
session.enableNotification(TdbNotificationType.GRAPH_ANALYSIS,true);

// To receieve actual path details, set the notification type
// on the TdbGraph object used. Omitting this will only return
// current path counts.

graph.setNotificationType(
    TdbGraphNotificationType.PATH_SNAPSHOT,
    true );
```

The TRIP Graph Query Language

TRIP implements a small graph query language tailored for TRIP's behavior and the way that graph support is implemented in TRIP. Using this query language, an application can perform several graph operations in sequence within the context of the same network transaction. All operations supported by the graph query language are possible to do programmatically, but such use will add performance overhead in terms of network traffic. This overhead is reduced when using the query language.

Syntax

A query STATEMENT is a list of STEPs, separated by the '/' character. A valid STATEMENT must contain at least one STEP. Some steps may only be found at the beginning of the statement, others may only be found at the end.

```
<STATEMENT> ::= <BEG-STEP> { "/" <MID-STEP> } [ "/" <END-STEP> ]
```

A query STEP can open a graph database (<GRAPH>), navigate forwards (<FOLLOW>), navigate backwards (<BACK>), search transitively (<TRANSITIVE>), search transitively in reverse (<REVERSE-TRANSITIVE>) resolve source vertices (<SOURCES>) and resolve target vertices (<TARGETS>).

```
<BEG-STEP> ::= <GRAPH> | <FOLLOW> | <BACK> | <TRANSITIVE> |  
               <REVERSE-TRANSITIVE>
```

```
<MID-STEP> ::= <FOLLOW> | <BACK> | <TRANSITIVE> |  
               <REVERSE-TRANSITIVE>
```

```
<END-STEP> ::= <SOURCES> | <TARGETS>
```

Each step is has the search generated by the previous step as context.

The GRAPH Step

The GRAPH step opens a graph database for search. This can only be put as the first step of the statement and can open a graph database, a permanent cluster of graph databases, or create a runtime cluster of graph databases.

If the GRAPH step is absent, the currently open database must be a graph database or a cluster of graph databases in order for the statement to execute successfully.

```
<GRAPH> ::= "GRAPH" "::" <IDENTIFIER>  
           | "GRAPH" "::" <CLUSTER-DEF>  
  
<CLUSTER-DEF> ::= <IDENTIFIER> "(" <IDENTIFIER> { "," <IDENTIFIER> } ")"
```

Example of opening a single graph database:

```
GRAPH::GRAPHBASE
```

Example of creating a runtime cluster of graph databases:

```
GRAPH::GRAPHCLUSTER(MAPS_DE,MAPS_SV,MAPS_FI)
```

The FOLLOW Step

The FOLLOW step does what the Follow method in TRIPjxp and TRIPnxp does. It navigates edges in the graph from source to target.

```
<FOLLOW> ::= "FOLLOW" "::" <EDGE-NAME> [ <FOLLOW-ARGS> ]  
  
<FOLLOW-ARGS> ::= "(" <SOURCE-SET> ")"  
                | "(" <SOURCE-SET> "," <TARGET-SET> ")"  
                | "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ")"
```


The BACK Step

The BACK step does what the Backtrack method in TRIPjxp and TRIPnpx does. It navigates edges in the graph from target to source.

```
<BACK> ::= "BACK" "::" <EDGE-NAME> [ <BACK-ARGS> ]

<BACK-ARGS> ::= "(" <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ")"
```

The TRANSITIVE Step

The TRANSITIVE step performs a forward-transitive search like what the Transitive method in TRIPjxp and TRIPnpx can do. Transitive search may be depth-limited and its result can be either all edges traversed or only the last set of edges traversed.

```
<TRANSITIVE> ::= "TRANSITIVE" "::" <EDGE-NAME> [ <TRANSITIVE-ARGS> ]

<TRANSITIVE-ARGS> ::= "(" <SOURCE-SET> ")"
| "(" <SOURCE-SET> "," <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ","
    <DEPTH> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ","
    <DEPTH> "," <ALLEDGES> ")"
```

The REVERSE-TRANSITIVE Step

The REVERSE-TRANSITIVE step performs a backwards-transitive search like what the Transitive method in TRIPjxp and TRIPnpx can do. Reverse transitive search may be depth-limited and its result can be either all edges traversed or only the last set of edges traversed.

```
<REVERSE-TRANSITIVE> ::= "REVERSE-TRANSITIVE" "::" <EDGE-NAME>
    [ <REVERSE-TRANSITIVE-ARGS> ]

<REVERSE-TRANSITIVE-ARGS> ::= "(" <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ","
    <DEPTH> ")"
| "(" <SOURCE-SET> "," <EDGE-SET> "," <TARGET-SET> ","
    <DEPTH> "," <ALLEDGES> ")"
```

The SOURCES Step

The SOURCES step can only be put at the very end of the query statement. It resolves a graph set of edges into a regular set with the records acting as source vertices in the edges of the graph set.

```
<SOURCES> ::= "SOURCES"
```

The TARGETS Step

The TARGETS step can only be put at the very end of the query statement. It resolves a graph set of edges into a regular set with the records acting as source vertices in the edges of the graph set.

```
<TARGETS> ::= "TARGETS"
```

Miscellaneous Syntax Entities

An edge name is either an identifier naming the edge type or an asterisk, meaning that all edges are valid:

```
<EDGE-NAME> ::= <IDENTIFIER> | "*"
```

Set numbers are integers:

<SOURCE-SET> ::= <INTEGER>

<TARGET-SET> ::= <INTEGER>

<EDGE-SET> ::= <INTEGER>

Depth information is an integer indicating how deep a transitive search may go.

<DEPTH> ::= <INTEGER>

The ALLEDGES value is an integer that acts as Boolean, indicating if all a set of all edges are to be returned from a transitive search or only the final set. Valid values are 0 for false and 1 for true.

<ALLEDGES> := "0" | "1"

Graph Query APIs

The Query method of the TdbGraphSet class is used in order to use the TRIP Graph Query Language.

Method: TdbGraphSet:Query

Type: TdbGraphQueryResult
Throws: TdbException

Java

```
TdbGraphQueryResult query(String statement);
```

.Net

```
TdbGraphQueryResult Query(String statement);
```

This method is called on the TdbGraphSet instance representing the total graph database edge set.

The query, if successful, returns an instance of TdbGraphQueryResult that can be used to explore the result.

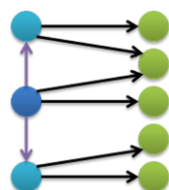
Class: TdbGraphQueryResult

Derived from: TdbQueryResult
Located in: graph

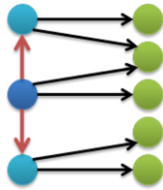
Examples

Shared Interests

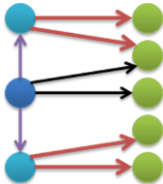
Finding out shared interests with ones friends. Assume a graph with the following structure. Blue vertices are individuals, green are interests, violet edges denote FRIEND relations and black edges denote INTEREST relations. Red is used to highlight the focus of the current operation.



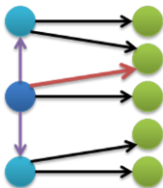
We start with the middle person and locate its friends:



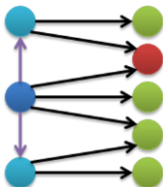
Then we find out what they are interested in:



Then we check if any of those are such that our starting person also are interested in:



Finally, we obtain the record(s) for the interest(s) we found:



In the TRIP Graph Query Language, we get the following expression. Set number 2 is assumed to contain the record for the individual we want to find shared interests for.

```
GRAPH::SOCIAL/FOLLOW::FRIEND(2)/FOLLOW::INTEREST/  
BACK::INTEREST(2)/TARGETS
```

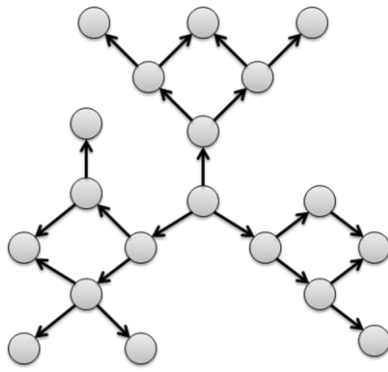
Extended Social Network

Locating all friends, their friends, and their friends as well. This will be a forward transitive search with a max depth of 3. Set number 2 is assumed to contain the record for the individual we want to find shared interests for.

```
GRAPH::SOCIAL/TRANSITIVE::FRIEND(2,0,0,3,1)/TARGETS
```

Here we pass zeroes for the edge and target sets. This indicates that we wish to impose no restriction on the edges and individuals we use in the search operation.

The set returned to the application would typically be used to display information like "you have 12345 people in your extended network", or to process this set further in order to find and display recent updates by these people.



The network we eventually find could have a topology like in the above illustration (centered on the starting person).

Populating a Graph

A TRIP graph database consists of records representing edges, i.e. relations. These relations are directional from a source vertex to a target vertex. Vertices are regular TRIP records; they do not have to be created specifically for use with the graph database. Any kind of record can be used as a graph vertex.

It is also possible to create records in a graph database that represent vertices. There are two main uses for this:

1. The vertices are very simple objects, usually with one identifying string.
2. Need for additional graph data about existing records without having to modify said records.

Create a Graph Database

Use TRIPmanager 7.1 or later to create a graph database. Select "graph" in the database type dropdown list.

New Database / Thesaurus

General Properties
Define the location of the database / thesaurus files

Files should be located using a logical name rather than a physical file path so that these files can be moved around at a later date without affecting the design.

Name: MYGRAPH

Database type: Graph

Available location names: LOCAL_BASES

Current mapping: D:\TripData\DB

☐ Use a transaction log file for backup / restore

Description: My graph database

< Back Next > Cancel Help

The graph database is a complete design with all fields required by TRIP to handle graph database, but it is also possible to add custom fields to a graph database if there is a need for it.

Adding an Edge

In order to add an edge record to a graph database, use the method `CreateEdge` on the `TdbGraph` class.

Method: `TdbGraph.CreateEdge`

Type: `TdbGraphPath`
Throws: `TdbException`

Java

```
TdbGraphRecord createEdge(String name, TdbRecord source,  
                           TdbRecord target, boolean commit);
```

.Net

```
TdbGraphRecord CreateEdge(String name, TdbRecord source,  
                           TdbRecord target, bool commit);
```

This method creates an edge record with a specified edge (relation) name between a source and a target record. The record is committed before return if the *commit* parameter is set to true. Set it to false in order to continue adding data to the record. If set to false, the application needs to call the `Commit` method on the returned record object.

Adding a Vertex

In order to add a vertex record to a graph database, use the method `CreateVertex` on the `TdbGraph` class.

Method: `TdbGraph.CreateVertex`

Type: `TdbGraphPath`
Throws: `TdbException`

Java

```
TdbGraphRecord createVertex(String label,  
                             TdbRecord reference,  
                             boolean commit);
```

.Net

```
TdbGraphRecord CreateVertex(String label,  
                             TdbRecord reference,  
                             bool commit);
```

This method creates a vertex record with a specified label. The label can be any value, but it is recommended that it is a unique value if the reference record is not set. The reference record will, if specified, associate the vertex with another record on the system. The record is committed before return if the *commit* parameter is set to true. Set it to false in order to continue adding data to the record. If set to false, the application needs to call the `Commit` method on the returned record object.

Modifying a Graph Record

The `TdbGraphRecord` class is used both in order to add further data to a record before it is committed for the first time, and to update it at a later stage. Instances of this class can be created directly using its constructors, or obtained via the `TdbRecordSet` and `TdbSearchSet` classes.

Class: TdbGraphRecord

Derived from:	<code>TdbRecord</code>
Located in:	<code>data</code>

The class contains methods for the manipulation of edges and vertices. Note that a graph record cannot change characteristics once it has been created. I.e. an edge must remain an edge and a vertex must remain a vertex. While circumventing this restriction is possible, it is NOT recommended because it will risk breaking the graph.



23. Cancelling Commands

In some circumstances it is necessary to be able to send a new request without having to wait for the previous one to finish. For example, a web application handling a field with search suggestions will cause rapid re-issuing of new versions of the search order. In such a situation, waiting for the previous order to complete is unnecessary since it will contain results that already are obsolete.

With TRIP 7.2 it is possible to cancel such commands from another thread.

To enable this functionality, first enable comforter notifications for the session object, passing the COMFORTER type and 'true' to enable to the EnableNotification method on the session object.

Method: TdbSession:EnableNotification

Type: void
Throws: TdbException

Java

```
session.enableNotification(TdbNotificationType, boolean);
```

.Net

```
Session.EnableNotification(TdbNotificationType, bool);
```

In order to be able to cancel a command, the property 'cancelable' needs to be set to true on the object responsible to the command before the command is executed.

Property: TdbSessionObject:Cancelable

Type: Boolean
Access: Read, Write

Java

```
boolean isCancelable();  
void setCancelable(boolean);
```

.Net

```
bool Cancelable { get; set; }
```

In order to cancel a running command, call the Cancel method on the object from another thread.

Method: TdbSessionObject:Cancel

Type: void
Throws: TdbException

Java

```
void cancel();
```

.Net

```
void Cancel();
```

For example:

Java

```
session.enableNotification(TdbNotificationType.COMFORTER,true);  
...  
TdbCclCommand cmd = TdbCclCommand(session);  
cmd.setCancelable(true);  
Thread cmdThread = new Thread(new Runnable() {  
    @Override  
    void run() {  
        cmd.execDirect(query);  
    }  
});  
cmdThread.start();  
...  
cmd.cancel();  
cmdThread.join();  
cmd.execDirect(newQuery());
```