Research is carried out by
NitrosData with grant support
from the Skolkovo Foundation

NitrosData
Link facts. Build knowledge.

NitrosBase RDF Storage C++ API

PROGRAMMER'S GUIDE

Overview **2**
Configuring the project to interact with NitrosBase RDF Storage **2**
Obtaining a database handle **2**
Opening a database connection **3**
Closing a database connection **3**
Adding data **3**
Preparing data in memory 3
Preparing data in a text file 4
Removing data **5**
Removing specified triples 5
Removing search query results 5
Removing data by the list of URIs 6
Modifying literal data **6**
Modyfying via removing and adding 6
Replacing data of a triple directly 7
Modifying referential data **7**
Backup and restore **7**
Retrieving data using SPARQL **8**
Executing a SPARQL query 8
Executing a SPARQL query with paging 8
Obtaining field count, names and types 8
Obtaining query results 9
Data obtain and output example 9
Functions list **11**
Functions reference **12**
Client application examples **21**

Overview

NitrosBase RDF Storage is a graph database based on W3C semantic standards, to include RDF and SPARQL. NitrosBase RDF Storage is designed with the purpose of combining the flexibility of RDF graph models with the power of high-performance databases.

This guide covers NitrosBase RDF Storage for 64 bit Linux and 64 bit Windows 7 / Windows server 2008 operating systems.

NitrosBase RDF Storage uses turtle/n3 for import/export and SPARQL (SPARQL Query Language for RDF / W3C Recommendation 15 January 2008 - http://www.w3.org/TR/rdf-sparql-query/) as a query language.

Configuring the project to interact with NitrosBase RDF Storage

To create a client C++ application, nitrosbase_rdf.h header file must be added to the project. It provides everything needed for opening database connections, executing queries, and retrieving the results.

Functions with string parameters or returning string values have two different implementations, one for strings in UTF8 encoding and one for strings in UTF16. UTF8 versions are preferable because UTF8 is used for internal representation. UTF16 versions are there to support programming languages containing UTF16 strings by default.

Note: all bundled code samples are pre-configured for proper database interaction.

Obtaining a database handle

To obtain a database handle, call NBGetDatabaseUTF8 (or NBGetDatabaseUTF16) function. For example, these calls:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", int len , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
```

or

```
int64_t dbh = NBGetDatabaseUTF16 ( "person", int len , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
```

will return a handle for the ‘person’ database.

Opening a database connection

To connect to the database, call NBConnect function:

```
int64_t connecth = NBConnect ( dbh );
```

This function returns a database connection handle that is used in all of the functions covered below.

Closing a database connection

To close a database connection, call NBCloseConnect function:

```
NBCloseConnect ( connecth );
```

Adding data

To add data, first prepare the data, and then call ExecuteQueryUTF8 (or ExecuteQueryUTF16) function. There are two possible ways to prepare the data:

1. In memory / string variable
2. In a text file

First option is preferable for small amounts of data, and the second one is better suited for significantly large amounts of data. In both cases, turtle/n3 format is used.

Preparing data in memory

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person10000> type k:Person. "
        "<http://Person10000> ID \"10000\"^^xsd:integer. "
        "<http://Person10000> First_Name \"Jonny\". "
        "<http://Person10000> Last_Name \"Fisher\". "
        "<http://Person10000> Male \"0\"^^xsd:integer. ";

ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTTRIPLES );
NBCloseConnect ( connecth );
```

or

```
int64_t dbh = NBGetDatabaseUTF16 ( L"person", 6 , L"c:\\NitrosBaseRDF\\data\\person\\",
L"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
wstring querytext =
        L"<http://Person10000> type k:Person. "
        L"<http://Person10000> ID \"10000\"^^xsd:integer. "
```

```
        L"<http://Person10000> First_Name \"Jonny\". "
        L"<http://Person10000> Last_Name \"Fisher\". "
        L"<http://Person10000> Male \"0\"^^xsd:integer. ";

ExecuteQueryUTF16 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTTRIPLES );
NBCloseConnect ( connecth );
```

In the example above **ExecuteQueryUTF8** (or **ExecuteQueryUTF16**) function uses **querytext** string (containing a set of turtle/n3 triples) as its parameter.

As the triples with subject of <http://Person10000> had not been added previously, they will be inserted as new data.

Preparing data in a text file

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string filepath = "c:\\NitrosBaseRDF\\data\\person\\person.n3";
ExecuteQueryUTF8 ( connecth, filepath.c_str(), filepath.length(), QUERY_INSERTTRIPLES_FILE );
NBCloseConnect ( connecth );
```

In this example, **filepath** is a path to a text file containing a set of triples in turtle/n3 format. Its contents could be the following:

```
<http://Person10000> type k:Person.
<http://Person10000> ID "10000"^^xsd:integer.
<http://Person10000> First_Name "Jonny".
<http://Person10000> Last_Name "Fisher".
<http://Person10000> Male "0"^^xsd:integer.
```

Warning!

Current NitrosBaseRDF version does not support multiple literal values of an object with identical subjects and predicates. For instance, you can't assign two different family names (objects) to a single person (subject). Therefore, the following triple arrangement is invalid:

```
<http://Person10000> Last_Name "Fisher".
<http://Person10000> Last_Name "Hunter".
```

However, this is possible with objects represented as URIs. For instance, an article can have several authors, and the following triple arrangement is valid:

```
<http://Article90000> Author <http://Person12345>.
<http://Article90000> Author <http://Person12345>.
```

This way of handling literal object values is typical for many RDF databases. We utilize this feature also as a means to implement a simple and elegant data modification algorithm (see "Modifying literal data" chapter).

Removing data

NitrosBase RDF Storage provides the following ways to remove data:

1. Removing specified triples.
2. Removing search query results.
3. Removing by the list of URIs.

Removing specified triples

Removes a set of triples contained in a string or a file.

The first option requires passing a set of triples as a string parameter:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person_clone19> System_Str_10p \"KKKK\". "
        "http://Person_clone19> p0link <http://Person4292>. ";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES );
NBCloseConnect ( connecth );
```

The second option reads a set of triples from a file, taking a full path to that file as a parameter:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string filepath = "c:\\NitrosBaseRDF\\data\\q_person\\deletetriples.n3";
ExecuteQueryUTF8 ( connecth, filepath.c_str(), filepath.length(), QUERY_DELETETRIPLES_FILE);
NBCloseConnect ( connecth );
```

In the example above, deletetriples.n3 file could contain the following lines:

```
<http://Person_clone19> System_Str_10p "KKKK".
<http://Person_clone19> p0link <http://Person4292>.
```

Removing search query results

Removes triples with URIs produced as a result of executing a search query. To achieve this, the client issues a query for a server to remove triples. This imposes a constraint on a query: it must be a SELECT statement with a list of SPARQL fields consisting of a single field having a URI type.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "SELECT ?Person "
        "WHERE  "
        "{ "
        "    ?Person ID ?ID1. "
```

```
        "   ?Person type k:Person. "
        "   FILTER(?ID1 < "6"^^xsd:integer) "
        "}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(),
        QUERY_DELETERECORDS_QUERY );
NBCloseConnect ( connecth );
```

In this example, all triples having URIs returned as a result of this query will be removed.

Removing data by the list of URIs

Removes all triples with URIs from the list.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person_clone19>\n"
        "<http://Person_clone33> \n"
        "<http://Person101>";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(),
        QUERY_DELETERECORDS_URILIST );
NBCloseConnect ( connecth );
```

In this example, all triples with URIs from **querytext** parameter will be removed.

Modifying literal data

There are two ways to modify existing data:

1. Removing and adding.
2. Direct replacement.

As in the "Adding data" chapter, either a memory string or a file can be a source of data. In both cases, turtle/n3 format is used.

Modyfying via removing and adding

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "<http://Person_clone19> System_Str_10p \"KKKK\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES);

querytext = "<http://Person_clone19> System_Str_10p \"JJJJ\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

First **ExecuteQueryUTF8** call with QUERY_DELETETRIPLES
removes the <http://Person_clone19> System_Str_10p "KKKK" triple.

Second **ExecuteQueryUTF8** call with QUERY_INSERTRIPLES
adds the <http://Person_clone19> System_Str_10p "JJJJ" triple.

Replacing data of a triple directly

This way of modifying data utilizes the fact that multiple object values in a triples having the same subject and predicate are not possible (see "Adding data" chapter). This allows for the replacement of an existing value by adding a triple with the same subject and predicate but different object.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
querytext = "<http://Person_clone19> System_Str_10p \"JJJJ\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

Since having multiple object literals with same subject and predicate is not possible, this **ExecuteQueryUTF8** call with QUERY_INSERTRIPLES parameter will overwrite "KKKK" with "JJJJ".

Modifying referential data

The only way to modify triples with URI as an object value is first removing an existing triple and then adding a new one:

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "<http://Person_clone11> p0link <http://Person9505>.";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES);

querytext = "<http://Person_clone11> p0link <http://Person9506>.";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

Backup and restore

To save a state of the database at the particular time, its data can be dumped into a file as a set of turtle/n3 triples.

Windows example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
```

```
string n3file = "c:\\NitrosBaseRDF\\data\\n3files\\person_19_05_2014.n3";
ExecuteQueryUTF8 ( connecth, n3file.c_str(), n3file.length(), DATABASE_SAVE2TURTLEN3 );
NBCloseConnect ( connecth );
```

Linux example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person" );
int64_t connecth = NBConnect ( dbh );
string n3file = "data/n3files/person_19_05_2014.n3";
ExecuteQueryUTF8 ( connecth, n3file.c_str(), n3file.length(), DATABASE_SAVE2TURTLEN3 );
NBCloseConnect ( connecth );
```

In this example database state as of May 19th, 2014 will be saved into a file of triples, 'person_19_05_2014.n3'.

To restore database state at that moment, just create a new database from that file.

Retrieving data using SPARQL

Executing a SPARQL query

ExecuteQueryUTF8 function with QUERY_SPARQL parameter issues a SPARQL query for the server to execute.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
NBCloseConnect ( connecth );
```

Executing a SPARQL query with paging

ExecuteQueryUTF8 function with QUERY_SPARQL parameter issues a SPARQL query for the server to execute, and retrieves a page of the result set.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
int count = ExecutePagingQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), from,  to,  afterto);
NBCloseConnect ( connecth );
```

Obtaining field count, names and types

To obtain field count, names, and types, the following functions are provided:

```
int         GetFieldCount ( int64_t connecth )
char *      GetFieldNameUTF8 ( int64_t connecth, int fieldnum, int * len )
```

```
wchar_t *    GetFieldNameUTF16 ( int64_t connecth, int fieldnum, int * len )
CLIENT_DATA_TYPE  GetFieldType ( int64_t connecth, int fieldnum )
```

Example:

```
int fcount = GetFieldCount ( connecth );
for ( int i = 0; i < fcount; i++ )
{
        int len;
        char * fieldname = GetFieldName (connecth, i, &len );
        CLIENT_DATA_TYPE fieldtype = GetFieldType (connecth, i );
        //… do something
}
```

Obtaining query results

A common intent after executing a query is to iterate through all retrieved records. The following example demonstrates summing all values of the third field (note that field indexing starts from 0):

```
int sum = 0;
while ( ReadRecord ( connecth ) )
{
   int * v = GetFieldInt ( connecth, 2 );
   if ( v != NULL )
      sum += *v;
}
```

After **ReadRecord** call, the following functions can be used to obtain field values for the current record:

```
int *        GetFieldInt ( int64_t connecth, int fieldnum )
int *        GetFieldInt64 ( int64_t connecth, int fieldnum ) // для получения значений sum(?x)
double *     GetFieldDouble ( int64_t connecth, int fieldnum )
char *       GetFieldDateTimeUTF8 ( int64_t connecth, int fieldnum,    int * len )
wchar_t *    GetFieldDateTimeUTF16 ( int64_t connecth, int fieldnum, int * len )
char *       GetFieldStringUTF8 ( int64_t connecth, int fieldnum,     int * len )
wchar_t *    GetFieldStringUTF16 ( int64_t connecth, int fieldnum,  int * len )
```

Data obtain and output example

The following example executes a SPARQL query and prints its results to the console in order to demonstrate the use of functions from this chapter:

NBConnectUTF8
ExecuteQueryUTF8
ExecutePagingQueryUTF8
ReadRecord
GetFieldCount
GetFieldNameUTF8
GetFieldType
GetFieldInt
GetFieldInt64
GetFieldDouble
GetFieldDateTimeUTF8

```
GetFieldStringUTF8
NBCloseConnect
try
{
	string querytext = "select ?x ?y ?z where{ ?x ?y ?z. }";
	ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
	while ( ReadRecord ( connecth ) )
	{
		int fcount = GetFieldCount ( connecth );
		for ( int i = 0; i < fcount; i++ )
		{
			char * fname = GetFieldNameUTF8 ( connecth, i, NULL );
			switch ( GetFieldType ( connecth, i ) )
			{
				case CLIENT_DATA_INT: {
					int * v = GetFieldInt ( connecth, i );
					if(v)
						cout << fname << " : " << *v << endl;
				}
				break;
				case CLIENT_DATA_INT64: {
					int64_t * v = GetFieldInt64 ( connecth, i );
					if(v)
						cout << fname << " : " << *v << endl;
				}
				break;
				case CLIENT_DATA_DOUBLE: {
					double * v = GetFieldDouble ( connecth, i );
					if(v)
						cout << fname << " : " << *v << endl;
				}
				break;
				case CLIENT_DATA_DATETIME: {
					char * v = GetFieldDateTimeUTF8 ( connecth, i, NULL );
					if(v)
						cout << fname << " : " << v << endl;
				}
				break;
				case CLIENT_DATA_URI:
				case CLIENT_DATA_STRING: {
					char * v = GetFieldStringUTF8 ( connecth, i, NULL );
					if(v)
						cout << fname << " : " << v << endl;
				}
				break;
			}
		}
	}
	NBCloseConnect ( connecth );
}
catch ( const char *error )
{
	cout << "Error : " << error << endl;
}
```

Functions list

Function	Purpose
NBConnectToDbUTF8 NBConnectToDbUTF16	Connect to the database named dbname and return a connection handle.
NBConnectOpen	Opens database connection.
NBCloseConnect	Closes database connection.
ExecuteQueryUTF8 ExecuteQueryUTF16	Perform a database query.
ExecutePagingQueryUTF8 ExecutePagingQueryUTF16	Perform a database query with paging
ReadRecord	Reads next query result record.
GetFieldCount	Returns the number of fields in current result set.
GetFieldNameUTF8 GetFieldNameUTF16	Return a name of designated field in current result set.
GetFieldType	Returns a type of designated field in current result set.
GetFieldInt	Returns a value of an int field in current result set.
GetFieldInt64	Returns a value of an int64 field in current result set.
GetFieldDouble	Returns a value of a double field in current result set..
GetFieldDateTimeUTF8 GetFieldDateTimeUTF16	Return a value of a datetime field in current result set.
GetFieldStringUTF8 GetFieldStringUTF16	Return a value of a string field in current result set.

Functions reference

NBGetDatabaseUTF8

Returns a handle for the database named dbname

Signature:

int64_t NBGetDatabaseUTF8 (const char * dbname, int len, const char * dbfolder, const char * nbserverfolder)

Parameters:

dbname – database name in the 8-bit encoding.

len – database name length.

dbfolder – database folder path.

nbserverfolder – nitrosbase db server module (and dlls) folder path.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\", "c:\\NitrosBaseRDF" );
```

NBGetDatabaseUTF16

Returns a handle for the database named dbname

Signature:

int64_t NBGetDatabaseUTF16 (const wchar_t * dbname, int len, const wchar_t * dbfolder, const wchar_t * nbserverfolder)

Parameters:

dbname – database name in the 16-bit encoding.

len – database name length.

dbfolder – database folder path.

nbserverfolder – nitrosbase db server module (and dlls) folder path.

Example:

```
int64_t dbh = NBGetDatabaseUTF16 ( L"person", 6 , L"c:\\NitrosBaseRDF\\data\\person\\", L"c:\\NitrosBaseRDF" );
```

NBConnect

Connects to the database and returns a connection handle

Signature:

Int64_t NBConnect (int64_t databasehandle)

Parameters:

databasehandle – database handle obtained with NBGetDatabaseUTF8 or NBGetDatabaseUTF16.

Return value:

database connection handle. A return value of 0 indicates a connection error.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
```

NBCloseConnect

Closes database connection

Signature:

void NBCloseConnect (int64_t connect, bool hardclose = false)

Parameters:

connecth – database connection handle.

hardclose – close mode; true will just close the connection, false will release its resources and place it into a connection pool where it can be reused upon subsequent NBConnect call.

Example:

```
NBCloseConnect ( connecth );
```

ExecuteQueryUTF8

Performs a database query.

Signature:

void ExecuteQueryUTF8 (int64_t connecth, const char * query, int length,
CLIENT_QUERY_TYPE qtype = QUERY_SPARQL)

Parameters:

connecth – database connection handle;

query – 8bit-encoded string that, depending on the type, consists either of query text, a set of triples, file name, and so on;

length – query length;

qtype – query type, one of the following:

```
enum CLIENT_QUERY_TYPE
{
	QUERY_SPARQL,
	QUERY_INSERTTRIPLES,
	QUERY_DELETETRIPLES,
	QUERY_INSERTTRIPLES_FILE,
	QUERY_DELETETRIPLES_FILE,
	QUERY_DELETERECORDS_QUERY,
	QUERY_DELETERECORDS_URILIST,
	QUERY_SAVE2TURTLEN3
};
```

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
NBCloseConnect ( connecth );
```

ExecuteQueryUTF16

Performs a database query.

Signature:

void ExecuteQueryUTF16 (int64_t connecth, const wchar_t * query, int length,
CLIENT_QUERY_TYPE qtype = QUERY_SPARQL)

Parameters:

connecth – database connection handle;

query – 16bit-encoded string that, depending on the type, consists either of query text, a set of triples, file name, and so on;

length – query length;

qtype – query type, one of the following:

```
enum CLIENT_QUERY_TYPE
{
	QUERY_SPARQL,
	QUERY_INSERTTRIPLES,
	QUERY_DELETETRIPLES,
	QUERY_INSERTTRIPLES_FILE,
	QUERY_DELETETRIPLES_FILE,
	QUERY_DELETERECORDS_QUERY,
	QUERY_DELETERECORDS_URILIST,
	QUERY_SAVE2TURTLEN3
};
```

ExecutePagingQueryUTF8

Performs a database query with paging.

Signature:

int ExecutePagingQueryUTF8 (int64_t connecth, const char * query, int length, int from, int to, int afterto)

Parameters:

connecth – database connection handle;

query – 8 bit-encoded string

length – query length;

from – first record of a page;

to – last record of a page;

afterto – one of the following:

afterto = MAX_INT — request exact number of records returned by this query;

afterto > 0 — request number of records after the page end;

afterto = 0 – ignored.

Example:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
int count = ExecutePagingQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), 0,  99,  1 );
NBCloseConnect ( connecth );
```

In this example, we are interested in a page of records starting with 0 and ending with 99, and a total number of records after this page, if any.

ExecutePagingQueryUTF16

Performs a database query with paging.

Signature:

int ExecutePagingQueryUTF16 (int64_t connecth, const wchar_t * query, int length, int from, int to, int afterto)

Parameters:

connecth – database connection handle;

query – 16 bit-encoded string;

length – query length;

from – first record of a page;

to – last record of a page;

afterto – one of the following

afterto = MAX_INT - request exact number of records returned by this query;

afterto > 0 - request number of records after the page end;

afterto = 0 – ignored.

ReadRecord

Reads next query result record

Signature:

bool ReadRecord (int64_t connecth)

Parameters:

connecth – database connection handle.

Return value:

true if there are more rows, false otherwise.

Example:

```
int sum = 0;
while ( ReadRecord ( connecth ) )
{
   int * v = GetFieldInt ( connecth, 2 );
   if ( v != NULL )
      sum += *v;
}
```

GetFieldCount

Returns the number of fields in the current result set.

Signature:

int GetFieldCount (int64_t connecth)

Parameters:

connecth – database connection handle.

Example:

```
Int fcount = GetFieldCount ( connecth );
```

GetFieldNameUTF8

Returns an 8 bit-encoded name of a designated field in the current result set.

Signature:

char * GetFieldNameUTF8 (int64_t connecth, int fieldnum, int * len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of a returned name; may be NULL if length is not needed.

Example:

```
int len;
char * fname = GetFieldNameUTF8 ( connecth, i, &len );
```

GetFieldNameUTF16

Returns a 16 bit-encoded name of a designated field in the current result set.

Signature:

wchar_t * GetFieldNameUTF16 (int64_t connecth, int fieldnum, int * len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of returned name; may be NULL if length is not needed.

Example:

```
int len;
wchar_t * fname = GetFieldNameUTF16 ( connecth, i, &len );
```

GetFieldType

Returns the type of designated field in the current result set, as one of the following values:

```
enum CLIENT_DATA_TYPE
{
	CLIENT_DATA_STRING,
	CLIENT_DATA_INT,
	CLIENT_DATA_INT64,
	CLIENT_DATA_DOUBLE,
	CLIENT_DATA_DATETIME,
	CLIENT_DATA_URI,
	CLIENT_DATA_VARTYPE
};
```

Signature

CLIENT_DATA_TYPE GetFieldType(int64_t connecth, int fieldnum)

Parameters:

connecth – database connection handle;

fieldnum – field index;

Example:

```
CLIENT_DATA_TYPE fieldtype = GetFieldType ( connecth, fieldnum );
```

GetFieldInt

Returns a value of an int field in the current result set.

Signature:

Int* GetFieldInt (int64_t connecth, int fieldnum)

Parameters:

connecth – database connection handle;

fieldnum – field index.

Return value:

Returns a pointer to an int if field is not NULL; returns NULL otherwise.

Example:

```
int * fieldvalue = GetFieldInt ( connecth, fieldnum );
```

GetFieldInt64

Returns a value of an int64_t field. **int64_t** type is not supported as a database type but is used for values of the **sum** aggregate function in SPARQL queries.

Signature:

int64_t* GetFieldInt64 (int64_t connecth, int fieldnum)

Parameters:

connecth – database connection handle;

fieldnum – field index.

Return value:

Returns a pointer to an int64_t if field is not NULL; returns NULL if otherwise.

Example:

```
int64_t * fieldvalue = GetFieldInt64 ( connecth, fieldnum );
```

GetFieldDouble

Returns a value of a double field. **double** type is supported as a database type, and it is also used for values of the **avg** aggregate function in SPARQL queries

Signature:

double* GetFieldDouble (int64_t connecth, int fieldnum)

Parameters:

connecth – database connection handle;

fieldnum – field index.

Return value:

Returns a pointer to a double if field is not NULL; returns NULL otherwise.

Example:

```
double * fieldvalue = GetFieldInt64 ( connecth, fieldnum );
```

GetFieldDateTimeUTF8

Returns a value of a datetime field.

Signature

char * GetFieldDateTimeUTF8 (int64_t connecth, int fieldnum, int * len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of the returned value; may be NULL if length is not needed..

Return value:

Returns a pointer to an 8 bit-encoded string if field is not NULL; returns NULL if otherwise.

Example:

```
int len;
char * fieldvalue = GetFieldDateTimeUTF8 ( connecth, fieldnum, &len );
```

GetFieldDateTimeUTF16

Returns a value of datetime field.

Signature:

wchar_t * GetFieldDatetimeUTF16 (int64_t connecth, int fieldnum, int* len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of the returned value; may be NULL if length is not needed.

Return value:

Returns a pointer to a 16 bit-encoded string if field is not NULL; returns NULL if otherwise.

Example:

```
int len;
wchar * fieldvalue = GetFieldDateTimeUTF16 ( connecth, fieldnum, &len );
```

GetFieldStringUTF8

Returns a value of a string field.

Signature:

char * GetFieldStringeUTF8 (int64_t connecth, int fieldnum, int* len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of the returned value; may be NULL if length is not needed.

Return value:

Returns a pointer to an 8 bit-encoded string if field is not NULL; returns NULL if otherwise.

Example:

```
int len;
char * fieldvalue = GetFieldStringUTF8 ( connecth, fieldnum, &len );
```

GetFieldStringUTF16

Returns a value of a string field.

Signature:

wchar_t * GetFieldStringUTF16 (int64_t connecth, int fieldnum, int* len)

Parameters:

connecth – database connection handle;

fieldnum – field index;

len – pointer to an int to hold the length of the returned value; may be NULL if length is not needed.

Return value:

Returns a pointer to a 16 bit-encoded string if field is not NULL; returns NULL if otherwise.

Example:

```
int len;
wchar_t * fieldvalue = GetFieldStringUTF16 ( connecth, fieldnum, &len );
```

Client application examples

NitrosBase RDF Storage API example applications may be found in C:\NitrosBaseRDF\Samples directory for Windows or in samples directory for Linux.

Sample1 – This is the most basic application example. It is a console application that connects to the *person* database, executes a simple SPARQL query, and prints its results.

Sample2 – This is an example of data manipulation (adding, removing and modifying data).

Sample3 – This is a basic application example to demonstrate paging. A console application that connects to the person database, executes a simple SPARQL query, and prints its results in several pages.