netgenium

External functions

Framework NET Genium

Content

1 Basic information

- The external function is used to call its own C# program code on the server side, which returns a value expressed by a text string.
- External functions are used in cases where
  - it is not possible to provide the required server – side functionality through a script, or
  - you need to call ajax requests from javascript.
- The server function “ngef(string id, string arg0, string arg1, string arg2, …)” is used to call external functions, which contains the required parameter “id” with the identifier of the external function, as well as any number of other optional parameters that are external to functions passed as “string[] args”.
- Errors or interruptions may occur while performing an external function.
  - External functions executed from the script in the event of an error or interrupt terminate the execution of the script, and return to the edit form or view page from which the script was called and display the error message to the user. For the “OnBeforeSave” and “OnBeforeDelete” scripts, such an interruption prevents the planned saving, resp. deleting a record.
  - External functions executed from the “HTML” or “JavaScript” control in the event of an error or interrupt stop the loading of the edit form or view page, and display an error message to the user.
  - External functions executed from other controls in the event of an error or interrupt stop data from loading into the control, and display an error message to the user directly in the control itself, without affecting the loading of other controls.

2 Library “ngef.dll”

- External functions are part of the “ngef.dll” library located in the “NETGenium\bin” directory. A clean installation of NET Genium has a default library “ngef.dll” in the “bin” directory, which is the result of compiling the default source code located in the “NETGenium\bin\ngef.cs” file.

```
using System;

namespace NETGenium
{
  public class ExternalFunctions
  {
    public static string ngef(string id, string[] args, bool test, DbCommand cmd, DbConnection
conn)
    {
      if (test) return "";

      switch (id)
      {
        // case "MyFirstFunction": return MyFirstFunction();
        default: return conn == null ? "" : conn.ExternalFunctionNotFound(id, cmd);
      }
    }
  }
}
```

- The source code for external functions uses objects and methods from the “NETGeniumConnection.dll” library, which is stored in the “NETGenium\bin” or “N:\NetGenium\Projekty\NetGenium\References” directory. “NETGeniumConnection.dll” is a library with basic functions for working with databases and file attachments.
- Modifications of the “ngef.dll” library can only be performed through a separate project in the “Visual Studio 2015” application and higher, resp. by programming the source code of this project, and then compiling the project into the library “ngef.dll”.
- A clean installation of NET Genium does not contain a project with source codes for external functions. Before you can program external functions, you must create a new library project in Visual Studio using the following steps, and add a reference to the “NETGeniumConnection.dll” library:
  - Start Visual Studio
  - From the menu on the main bar, select “File/New/Project…” (Ctrl+Shift+N)
    - Project type: Class Library (.NET Framework)
    - Project name: ngef
    - Location: optional project location
    - Solution: Create new solution
    - Place solution and project in the same directory: Yes
    - Create directory for solution: No (Visual Studio 2015)
    - Framework: .NET Framework 4.5.2
  - Right-click on the file “Class1.cs”, and select “Delete”

- Right-click on the project name “ngef”, select “Add” / “Existing Item…” (Shift+Alt+A), and select the path to the file “NETGenium\bin\ngef.cs” on the computer disk
- Right-click on “References”, select “Add Reference…”, and select the path to the file “References\NETGeniumConnection.dll” on the computer disk
- Right-click on “References” / “NETGeniumConnection”, select “Properties” (Alt+Enter), and set the “Copy Local” attribute to “False”
- Choose “Debug” compilation mode
  - “Debug” mode generates “ngef.dll” and “ngef.pdb” files by default
  - Thanks to the “ngef.pdb” file, errors and interrupts in external functions are easily detected, because the “Stack Trace” of each error also includes the file name and the line number on which the interrupt occurred.
  - The “Release” mode is only recommended for the final version of the tuned source code in the “ngef.dll” library. By default, the “Release” mode only generates the “ngef.dll” file, which is sufficient for NET Genium, but when interrupted, finding the cause of the error is very complicated.
- Compile the project – from the menu on the main bar select “Build” / “Build Solution” (Ctrl+Shift+B)
- Copy the files “ngef.dll” and “ngef.pdb” from the directory “bin\Debug” to the directory “NETGenium\bin” if the library is compiled in “Debug” mode, or the file “ngef.dll” from the directory “bin\Release”) to the “NETGenium\bin” directory when compiling the “ngef.dll” library in “Release” mode. In the case of “Release” mode, it is important to delete the “ngef.pdb” file from the “NETGenium\bin” directory.
  - Uploading a new version of the “ngef.dll” library will always restart the web application, as will any change in the “NETGenium\bin” directory.

- The source code of external functions can be freely modified, but the following conventions in the “ngef.cs” file must not be changed:
  - Namespace NETGenium
  - public class ExternalFunctions
  - public static string ngef(string id, string[] args, bool test, DbCommand cmd, DbConnection conn)
  - if (test) return "";
  - default: return conn == null? "": conn.ExternalFunctionNotFound(id, cmd);

3 Parameters for the “public static string ngef” function

- string id
  - An external function identifier that uniquely identifies a function or program code executed by calling the server function “ngef(id)”.
  - It is necessary to create a separate “case” for each identifier inside the “switch” command in the “public static string ngef” function guide.
- string[] args
  - External function parameters that are part of the server function call “ngef(id, arg0, arg1, arg2,…)” in the second and next position in the parameter list.
  - The “args” parameter list can have 0 elements if the call to the “ngef(id)” server function contains only the identifier of the external function without additional parameters.
- bool test
  - The logical value “test” determines whether the external function is called from the script designer using the “Run script” button.
  - The default source code for external functions includes an “if(test) return "";” statement on the first line of the “public static string ngef” function, which ensures that the external function from the script designer does not run unintentionally when debugging the script.
- DbCommand cmd
  - The “cmd” object specifies a database object of the “DbCommand” type, which is used to write data to the database.
  - External functions executed from a script using the “ngef” server function, together with other script commands, use a single database object “cmd”, which is used to write or delete data from the database in a single transaction of type “IsolationLevel.ReadCommitted”.
  - Database operations performed via the “cmd” object do not take effect in the database until the transaction is committed. This occurs automatically at the end of each successful script, or by calling the server function “COMMIT()”.
  - Any error or interruption during the execution of the external function or the script itself will ensure a “rollback” of all database operations performed via the “cmd” object.
  - The use of the “cmd” object is not appropriate in cases of bulk data imports, or in general in cases where data entry is not required within the transaction. Typical data changes through the “cmd” object contain at most units of write or delete commands.
  - Using the “cmd” object can cause a “deadlock” of the database. When working with an object, it is always important to load everything needed from the database first, and only then write it to the database. Deadlock arises in situations where the programmer first writes data to a database table and then tries to read from the same database table.
    - For a Firebird database, a “deadlock” occurs when trying to read from the database table to which the transaction was written. Thus, Firebird locks the entire database table to which it was written during the transaction.

        - For an MSSQL database, a “deadlock” occurs when trying to read from a row of the database table to which the transaction was written. Therefore, MSSQL locks the rows of the database table to which it was written during the transaction.
    - Whenever it is not necessary to write to the database within a single transaction together with a script, it is recommended to use your own object “cmd” – for example using the command “using (DbCommand cmd = new DbCommand(conn)) {}”.
    - An external function executed from a location other than the script has the “cmd” object set to “null”.
- DbConnection conn
    - The “conn” object specifies a database object of the “DbConnection” type, which represents a connection to the NET Genium database, and is used to read or write data to the database.

4 Debugging external functions in the console application

- The most convenient way to write external functions is to design a prototype of an external function in a console application, and then copy the tuned source code to the "ngef.dll" library project.
- Compiling and running a console application is very fast, and allows you to easily debug and trace either using "breakpoints" or by listing information to the console using the "Console.WriteLine()" command.
- When designing an external function, it is important to choose a new external function declaration so that the resulting source code can be easily transferred to the "ngef.dll" library project using "Ctrl+C" and "Ctrl+V".

4.1 Creating a console application

- In the first step, you need to create a new console application project in Visual Studio using the following steps, and add a reference to the "NETGeniumConnection.dll" library:
    - Start Visual Studio
    - From the menu on the main bar, select "File / New / Project…" (Ctrl+Shift+N)
        - Framework: .NET Framework 4.5.2
        - Project Type: Console Application
        - Name: ConsoleApplication1
        - Location: optional project location
        - Solution: Create new solution
        - Create directory for solution: No
        - Add to Source Control: No
    - Right-click on "References", select "Add Reference…", and select the path to the file "NETGenium\bin\NETGeniumConnection.dll" on the computer disk
    - Choose "Debug" compilation mode
    - Start project – select "Debug" / "Start Debugging" (F5) from the menu on the main bar

4.2 NET Genium environment simulation

- In the second step, you must modify the default source code in the “Program.cs” file so that the console application simulates the NET Genium environment, which runs external functions by calling “public static string ngef” with the parameters “args”, “cmd” and “conn”.

```
using NETGenium;
using System;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      NETGeniumConsole console = new NETGeniumConsole();

      using (DbConnection conn = new
DbConnection(@"driver=firebird;datasource=localhost;user=SYSDBA;password=masterkey;database=C:
\Firebird\netgenium.fdb;charset=WIN1250;collation=WIN_CZ"))
      // using (DbConnection conn = new
DbConnection("server=(local);Trusted_Connection=true;database=netgenium"))
      using (DbCommand cmd = new DbCommand(conn))
      {
        conn.Open();

        conn.RootPath = "C:\\inetpub\\wwwroot\\netgenium";
        DateTime now = DateTime.Now;

        Console.WriteLine(MyFirstFunction(new string[] { "a", "b", "c" }, cmd, conn));

        Console.WriteLine();
        Console.WriteLine(conn.User.FormatTimeSpan(DateTime.Now - now));
      }

      console.Exit();
    }

    private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
    {
      return "Hello World! args: " + string.Join(";", args);
    }
  }
}
```

4.3 Writing the source code of an external function

- In the third step, it is then possible to write the source code of the external function “MyFirstFunction”.

5 Reading data from the database

5.1 Retrieve records from an SQL query into a DataTable object

```
using NETGenium;
using System;
using System.Data;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DataTable data = Data.Get("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
  Console.WriteLine(conn.User.FormatDataTableText(data));

  foreach (DataRow row in data.Rows)
  {
    // Console.WriteLine(row["id"]);
  }

  return "";
}
```

5.2 Retrieve records from an SQL query into a DbRow object

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DbRow row = new DbRow("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (row.Read())
  {
    // Console.WriteLine(row["id"]);
    Console.WriteLine(row.Report());
  }

  return "";
}
```

5.3 Value parsing

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DbRow row = new DbRow("SELECT * FROM sholiday WHERE date_ > " + conn.Format(new
DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (row.Read())
  {
    int id = (int)row["id"];
    Console.WriteLine("id: " + id);

    int pid = Parser.ToInt32(row["pid"]);
    Console.WriteLine("pid: " + pid);

    double _pid = Parser.ToDouble(row["pid"]);
    Console.WriteLine("pid: " + _pid);

    string name = row["name"].ToString();
    Console.WriteLine("name: " + name);

    DateTime date = Parser.ToDateTime(row["date_"]);
    Console.WriteLine("date: " + conn.User.FormatDateTime(date));
  }

  return "";
}
```

5.4 Indexing of retrieved records by primary key

```
using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DataTable data = Data.Get("SELECT * FROM sholiday", conn);
  Dictionary<int, DataRow> dictionary = Data.DictionaryInt32(data);

  int id = 1;
  if (dictionary.ContainsKey(id))
  {
    DataRow row = dictionary[id];
    Console.WriteLine(row["id"]);
  }

  return "";
}
```

5.5 Indexing of retrieved nested records according to the foreign key “pid”

```
using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DataTable data = Data.Get("SELECT * FROM sholiday", conn);
  Dictionary<int, List<DataRow>> dictionary = Data.DictionaryInt32(data, "pid", true);

  int pid = 0;
  if (dictionary.ContainsKey(pid))
  {
    List<DataRow> rows = dictionary[pid];
    Console.WriteLine(rows.Count + " rows");
  }

  return "";
}
```

6 Writing data to the database

6.1 INSERT INTO – create a new record in the database

6.1.1 DataSaver – create one record

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DataSaver ds = new DataSaver("sholiday", 0, cmd);
  ds.Add("name", "Nový rok");
  ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
  ds.Execute();

  Console.WriteLine(ds.Report());

  return "";
}
```

6.1.2 DataSaver – creation of two records in a separate transaction

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  cmd.Transaction = conn.BeginTransaction();

  try
  {
    DataSaver ds = new DataSaver("sholiday", 0, cmd);
    ds.Add("name", "Nový rok");
    ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
    ds.Execute();

    Console.WriteLine(ds.Report());

    ds = new DataSaver("sholiday", 0, cmd);
    ds.Add("name", "Nový rok");
    ds.Add("date_", new DateTime(DateTime.Today.Year + 1, 1, 1));
    ds.Execute();

    Console.WriteLine(ds.Report());

    cmd.Transaction.Commit();
  }
  catch (Exception ex)
  {
    cmd.Transaction.Rollback();
    throw ex;
  }
```

```
  return "";
}
```

6.1.3 DataSaver – create one record by copying another record

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = 1;

  DbRow row = new DbRow("SELECT * FROM sholiday WHERE id = " + id, conn);
  if (row.Read())
  {
    // row["ng_zadanokym"] = conn.User.LoginName;
    // row["ng_zadanokdy"] = DateTime.Now;
    // row["ng_zmenenokym"] = DBNull.Value;
    // row["ng_zmenenokdy"] = DBNull.Value;

    DataSaver ds = new DataSaver("sholiday", 0, cmd);
    ds.Add(row);
    ds.Execute();

    Console.WriteLine(ds.Report());
  }

  return "";
}
```

6.1.4 DataSaverSynchro – creation of one record, including creation of history record and ensuring record synchronization

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  DataSaverSynchro ds = new DataSaverSynchro("sholiday", 0, conn);
  ds.Add("name", "Nový rok");
  ds.Add("date_", new DateTime(DateTime.Today.Year, 1, 1));
  ds.Save(cmd);

  Console.WriteLine(ds.Report());

  return "";
}
```

6.2 UPDATE – editing an existing record in the database

6.2.1 DataSaver – editing a record whose ID is retrieved from the database

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový
rok") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (id != 0)
  {
    DataSaver ds = new DataSaver("sholiday", id, cmd);
    ds.Add("name", "Nový rok - TEST");
    ds.Execute();

    Console.WriteLine(ds.Report());
  }

  return "";
}
```

6.2.2 DataSaver – editing a record whose ID is stored in a variable; if the record does not exist, it will be created

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = 1;

  DataSaver ds = new DataSaver("sholiday", id, true, cmd);
  ds.Add("name", "Nový rok - TEST");
  ds.Execute();

  Console.WriteLine(ds.Report());

  return "";
}
```

6.2.3 DataSaverSynchro – editing a record whose ID is retrieved from the database, including creating a history record and ensuring record synchronization

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový
rok") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (id != 0)
  {
    DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, conn);
    ds.Add("name", "Nový rok - TEST");
    ds.Save(cmd);

    Console.WriteLine(ds.Report());
  }

  return "";
}
```

6.2.4 DataSaverSynchro – editing a record whose ID is stored in a variable, including creating a history record and ensuring record synchronization; if the record does not exist, it will be created

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = 1;

  DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, true, conn);
  ds.Add("name", "Nový rok - TEST");
  ds.Save(cmd);

  Console.WriteLine(ds.Report());

  return "";
}
```

6.3 Data synchronization of two database tables according to a single key

```
using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  string key = "ng_osobnicislo";

  DataTable data1 = Data.Get("SELECT * FROM ng_data1", conn), data2 = Data.Get("SELECT * FROM
ng_data2", conn);
  Dictionary<int, DataRow> dictionary = Data.DictionaryInt32(data1);

  foreach (DataRow row2 in data2.Rows)
  {
    DataRow row1 = Data.DataRow(dictionary, Parser.ToInt32(row2[key]));
    if (row1 == null)
    {
      DataSaver ds = new DataSaver("ng_data1", 0, cmd);

      for (int i = 1; i < data2.Columns.Count; i++)
      {
        ds.Add(data2.Columns[i].ColumnName, row2[data2.Columns[i].ColumnName]);
      }

      ds.Execute();
    }
    else
    {
      DataSaver ds = new DataSaver("ng_data1", (int)row1["id"], cmd);
      for (int i = 1; i < data2.Columns.Count; i++)
        if (row1[data2.Columns[i].ColumnName].ToString() !=
row2[data2.Columns[i].ColumnName].ToString())
        {
          ds.Add(data2.Columns[i].ColumnName, row2[data2.Columns[i].ColumnName]);
        }

      if (!ds.Empty)
      {
        ds.Execute();
      }
    }
  }

  return "";
}
```

7 Deleting data from the database

7.1 DELETE FROM – delete a record from the database

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok
- TEST") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (id != 0)
  {
    cmd.CommandText = "DELETE FROM sholiday WHERE id = " + id;
    cmd.ExecuteNonQuery();
  }

  return "";
}
```

7.2 DataSaverSynchro – deleting a record from the database, including creating a history record and ensuring record synchronization

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = Data.ExecuteScalar2("SELECT id FROM sholiday WHERE name = " + conn.Format("Nový rok
- TEST") + " AND date_ = " + conn.Format(new DateTime(DateTime.Today.Year, 1, 1)), conn);
  if (id != 0)
  {
    DataSaverSynchro ds = new DataSaverSynchro("sholiday", id, conn);
    ds.Delete(cmd);
  }

  return "";
}
```

8 Edit forms

8.1 Find out the ID of the currently open edit form

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  string dbname = conn.FormData.Table.TableName;
  int form = Parser.ToInt32(Config.QueryString("form"));

  return "";
}
```

8.2 Retrieve the value of a control on an edit form

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  string name = conn["name"].ToString();

  return "";
}
```

8.3 Save the value to a control in an edit form

```
using NETGenium;
using System;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  conn["name"] = "Nový rok - TEST";

  return "";
}
```

8.4 Ajax call from javascript

8.4.1 JavaScript

```
var p0 = 'ěščřžýáié', p1 = 'ĚŠČŘŽÝÁÍÉ';
loadUrl('ngef.aspx?MyFirstFunction,' + urlEncode(p0) + ',' + urlEncode(p1), 'ajaxResponse',
'test', 'p0=' + urlEncode(p0) + '&p1=' + urlEncode(p1));

function ajaxResponse(html)
{
  alert(html);
}
```

8.4.2 External functions

```
case "MyFirstFunction": MyFirstFunction(args, conn); return "";

using NETGenium;
using System;

private static void MyFirstFunction(string[] args, DbConnection conn)
{
  StringBuilder sb = new StringBuilder();
  sb.Append("MyFirstFunction REPORT");
  sb.Append(" | GET args: ");
  sb.Append(string.Join(", ", args));
  sb.Append(" | POST args: ");
  sb.Append(conn.Page.Request.Form["p0"]);
  sb.Append(", ");
  sb.Append(conn.Page.Request.Form["p1"]);
  Html.FlushAjaxContent(sb.ToString(), conn);
}
```

8.5 Simulation of opening a specific record in an edit form from a console application and reading the value of a control

```
using NETGenium;
using System;
using System.Data;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  int id = 1;

  DataTable data = Data.Get("SELECT * FROM sholiday WHERE id = " + id, conn);
  if (data.Rows.Count != 0)
  {
    conn.Register(data.Rows[0]);
  }

  string name = conn["name"].ToString();
  Console.WriteLine(name);

  return "";
}
```

8.6 Capture events in the edit form

8.6.1 Open the edit form

```
// case "NETGenium.OnAfterOpen": OnAfterOpen(args, conn); return "";

using NETGenium;
using System;

private static void OnAfterOpen(string[] args, DbConnection conn)
{
  int form = Parser.ToInt32(args[0]);
}
```

8.6.2 Save the record

```
// case "NETGenium.OnAfterSave": OnAfterSave(args, conn); return "";

using NETGenium;
using System;

private static void OnAfterSave(string[] args, DbConnection conn)
{
  int form = Parser.ToInt32(args[0]), id = (int)conn["id"];
}
```

8.6.3 Deleting a record

```
// case "NETGenium.OnAfterDelete": OnAfterDelete(args, conn); return "";

using NETGenium;
using System;

private static void OnAfterDelete(string[] args, DbConnection conn)
{
  int form = Parser.ToInt32(args[0]), id = (int)conn["id"];
}
```

9 View tables

9.1 Execute an external function with “ngef2” from the lookup table

9.1.1 Edits in NET Genium

- Create a new textbox in the “User” form
  - Name: “Test”
  - Check “Read Only”
  - Check “Hidden field”
  - Default value: ngef2(ngef2test)
  - Check “Fill in default value every time the edit form is opened”
- Create a new preview page with a datagrid that will only display the “Test” column

9.1.2 Edits in external function

```
// case "ngef2test": return ngef2test(args, conn);

using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static string ngef2test(string[] args, DbConnection conn)
{
  int id = Parser.ToInt32(args[args.Length - 1]);

  string key = "ngef2test";
  Dictionary<int, DataRow> dictionary;

  if (!conn.Container2.ContainsKey(key))
  {
    DataTable data = Data.Get("SELECT id, loginname FROM susers", conn);
    dictionary = Data.DictionaryInt32(data);
    conn.Container2.Add(key, dictionary);
  }
  else
  {
    dictionary = (Dictionary<int, DataRow>)conn.Container2[key];
  }

  if (dictionary.ContainsKey(id))
  {
    return dictionary[id]["id"] + ": " +
      dictionary[id]["loginname"].ToString();
  }

  return id.ToString();
}
```

9.2 Filling in the values in the view table

9.2.1 Edits in NET Genium

- Create a new textbox in the "User" form
  - Name: "Test"
- Find out the textbox ID: for example "7125"
- Create a new preview page with a datagrid that will only display the "Test" column
  - Check "ngef(NETGenium.DataTable)" in the data source on the "Other" tab
  - Copy the source code sample to the cliboard under the check box:
    - if (args[0] == "Q987" && args[1] == "1")
    - {
    - DataTable data = (DataTable)conn.Container2[args [0]];
    - int form = Parser.ToInt32(args [1]);
    - }

9.2.2 Edits in external function – version 1 for a small number of records

```
// case "NETGenium.DataTable": QueryBuilder(args, conn); return "";

using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static void QueryBuilder(string[] args, DbConnection conn)
{
  if (args[0] == "Q987" && args[1] == "1")
  {
    DataTable data = (DataTable)conn.Container2[args[0]];
    int form = Parser.ToInt32(args[1]);

    string key = "c7125";
    if (data.Columns.Contains(key) && data.Rows.Count != 0)
    {
      List<int> ids = new List<int>();

      foreach (DataRow row in data.Rows)
      {
        int id = (int)row["id"];
        if (id != 0)
        {
          ids.Add(id);
        }
      }

      DataTable data2 = Data.Get("SELECT id, loginname FROM susers WHERE id IN (" +
ParserSql.Ids(ids.ToArray()) + ")", conn);
      Dictionary<int, DataRow> dictionary = Data.DictionaryInt32(data2);

      foreach (DataRow row in data.Rows)
      {
```

```
        int id = (int)row["id"];
        if (dictionary.ContainsKey(id))
        {
          row[key] = dictionary[id]["id"] + ": " +
            dictionary[id]["loginname"].ToString();
        }
        else
        {
          row[key] = dictionary[id]["id"].ToString();
        }
      }
    }
  }
}
```

9.2.3 Edits in external function – version 2 for a large number of records

```
// case "NETGenium.DataTable": QueryBuilder(args, conn); return "";

using NETGenium;
using System;
using System.Collections.Generic;
using System.Data;

private static void QueryBuilder(string[] args, DbConnection conn)
{
  if (args[0] == "Q987" && args[1] == "1")
  {
    DataTable data = (DataTable)conn.Container2[args[0]];
    int form = Parser.ToInt32(args[1]);

    string key = "c7125";
    if (data.Columns.Contains(key) && data.Rows.Count != 0)
    {
      Dictionary<int, DataRow> dictionary = new Dictionary<int, DataRow>();
      List<int> ids = new List<int>();

      foreach (DataRow row in data.Rows)
      {
        int id = (int)row["id"];
        if (id != 0 && !dictionary.ContainsKey(id))
        {
          dictionary.Add(id, row);
          ids.Add(id);
        }
      }

      if (ids.Count != 0)
        foreach (string group in ParserSql.IdsInGroupsOf100(ids.ToArray()))
        {
          DbCommand cmd = new DbCommand("SELECT id, loginname FROM susers WHERE id IN (" +
group + ")", conn);
          DbDataReader dr = cmd.ExecuteReader();
          while (dr.Read())
          {
```

```
            int id = (int)dr["id"];
            if (dictionary.ContainsKey(id))
            {
              dictionary[id][key] = id + ": " + dr["loginname"].ToString();
            }
          }

          dr.Close();
          cmd.Dispose();
        }
    }
  }
}
```

9.3 Export data from a statistical look-up table

```
// case "NETGenium.DataGrid2Excel": DataGrid2Excel(args, conn); return "";

using NETGenium;
using System;

private static void DataGrid2Excel(string[] args, DbConnection conn)
{
  string path = args[0], query = args[1];
  int form = Parser.ToInt32(args[2]);
  DateTime date1 = Parser.ToDateTime(args[3]);
  DateTime date2 = Parser.ToDateTime(args[4]);
}
```

10 File attachments

10.1 Creating a file attachment

```
using NETGenium;
using System;
using System.IO;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  string temp = Path.GetTempFileName();
  Files.Write(temp, "abc");

  int file = Attachment.Add("test.txt", temp, conn);
  File.Delete(temp);
  Console.WriteLine(file);

  return "";
}
```

10.2 Load the contents of a file attachment

```
using NETGenium;
using System;
using System.IO;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  string temp = Path.GetTempFileName();
  Files.Write(temp, "abc");

  int file = Attachment.Add("test.txt", temp, conn);
  File.Delete(temp);
  Console.WriteLine(file);

  string path = Attachment.FilePath(file, conn);
  if (File.Exists(path))
  {
    string s = File.ReadAllText(path);
    Console.WriteLine(s);
  }

  return "";
}
```

10.3 Prevent downloading of a file attachment – exchange of content for an empty file

```
// case "NETGenium.Download": return Download(args, conn);

using NETGenium;
using System;

private static string Download(string[] args, DbConnection conn)
{
  int id = Parser.ToInt32(args[0]);
  string filename = args[1], path = args[2];

  L.N("Download: " + conn.User.LoginName + "; " + id + "; " + filename + "; " + path);

  path = conn.RootPath + "Images\\1x1.gif";
  return "OK\r\n" + path;
}
```

11 E-mails

11.1 Sending an e-mail message

```
using NETGenium;
using System;
using System.Net.Mail;

private static string SendMessage(string[] args, DbConnection conn)
{
  string html = NETGenium.Email.Message.Container("<b>Hello</b>");

  MailMessage message = new MailMessage();
  message.From = new MailAddress("@");
  message.To.Add(new MailAddress("@"));
  message.Subject = "";
  message.AlternateViews.Add(Config.CreateTextAlternateView(Html.ToText(html)));
  message.AlternateViews.Add(Config.CreateHtmlAlternateView(html, conn));

  Config.SendMessage(message, conn);
}
```

11.2 Capture the event of sending an e-mail message via the “New e-mail” form, and skip saving the message to the sent ones

```
// case "NETGenium.SendMessage": return SendMessage(args, conn);

using NETGenium;
using System;

private static string SendMessage(string[] args, DbConnection conn)
{
  MailMessage message = null;
  bool save = false;

  foreach (object co in conn.Container)
    if (co is MailMessage)
    {
      message = (MailMessage)co;
    }
    else if (co is bool)
    {
      save = (bool)co;
    }

  return "skipsave";
}
```

12 Printing to printing templates

12.1 Capture a print event in a print template, and change the contents or name of the printed file

```
// case "NETGenium.Print": Print(conn); return "";

using NETGenium;
using System;

private static void Print(DbConnection conn)
{
  if (conn.PrintingProcess.Template == "Test.pdf")
  {
    string path = conn.PrintingProcess.FilePath;
    Files.Write(path, "abc");
    conn.PrintingProcess.FileName = "abc.txt";
  }
}
```

12.2 Change the password for locking Excel print templates

```
// case "NETGenium.ExcelPassword": return ExcelPassword();

using NETGenium;
using System;

private static string ExcelPassword()
{
  return "OK\r\n" + Guid.NewGuid().ToString();
}
```

13 User login

13.1 Capture the “OnBeforeLogin” event immediately before the user automatically logs on through Active Directory

```
// case "NETGenium.OnBeforeLogin": return OnBeforeLogin(args, conn);

using NETGenium;
using System;

private static void OnBeforeLogin(string[] args, DbConnection conn)
{
  string account = args[0];
}
```

13.2 Prevent users from logging on

```
// case "NETGenium.Login": return Login(args, conn);

using NETGenium;
using System;

private static string Login(DbConnection conn)
{
  if (HttpContext.Current.Request.UserHostAddress == "127.0.0.1")
  {
    return "";
  }

  HttpContext.Current.Session.Abandon();

  return "OK\r\nalert('Invalid login.'); if (opener != null) window.close(); else top.location
= 'Default.aspx';";
}
```

14 Other

14.1 Logging to disk in the “Logs” directory

```
using NETGenium;
using System;
using System.IO;

private static string MyFirstFunction(string[] args, DbCommand cmd, DbConnection conn)
{
  try
  {
    throw new NullReferenceException("args");
  }
  catch (Exception ex)
  {
    L.E("MyFirstFunction", ex);
    // L.LogError("MyFirstFunction", ex);
    // L.Error("MyFirstFunction", ex);

    L.W("MyFirstFunction", ex);
    // L.LogWarning("MyFirstFunction", ex);
    // L.Warning("MyFirstFunction", ex);

    L.N("MyFirstFunction", ex);
    // L.LogNotice("MyFirstFunction", ex);
    // L.Notice("MyFirstFunction", ex);
  }

  return "";
}
```

14.2 Capture a database record lock event

```
// case "NETGenium.LockRecord": LockRecord(); return "";
// case "NETGenium.UnlockRecord": UnlockRecord(); return "";

using NETGenium;
using System;

private static void LockRecord(string[] args, DbConnection conn)
{
  int form = Parser.ToInt32(args[0]);
  long id = Parser.ToInt64(args[1]);
}

private static void UnlockRecord(string[] args, DbConnection conn)
{
  int form = Parser.ToInt32(args[0]);
  long id = Parser.ToInt64(args[1]);
}
```

14.3 Change the content of the text “Copyright”

```
// case "NETGenium.Copyright": return Copyright(conn);

using NETGenium;
using System;

private static string Copyright(DbConnection conn)
{
  return "OK\r\n © NetGenium " + DateTime.Now.Year;
}
```

14.4 Additional CSS style adjustments when saving the skin

```
// case "NETGenium.CSS": return CSS(args);

using NETGenium;
using System;

private static string CSS(string[] args)
{
  string value = args[0], browser = args[1];
  return "OK\r\n" + value + ".teststyle { color: red; }";
}
```