



Cognitier SimpleIPC

Exercise 3 – Object Pooling

Version 1.0.0.1

April 21, 2009

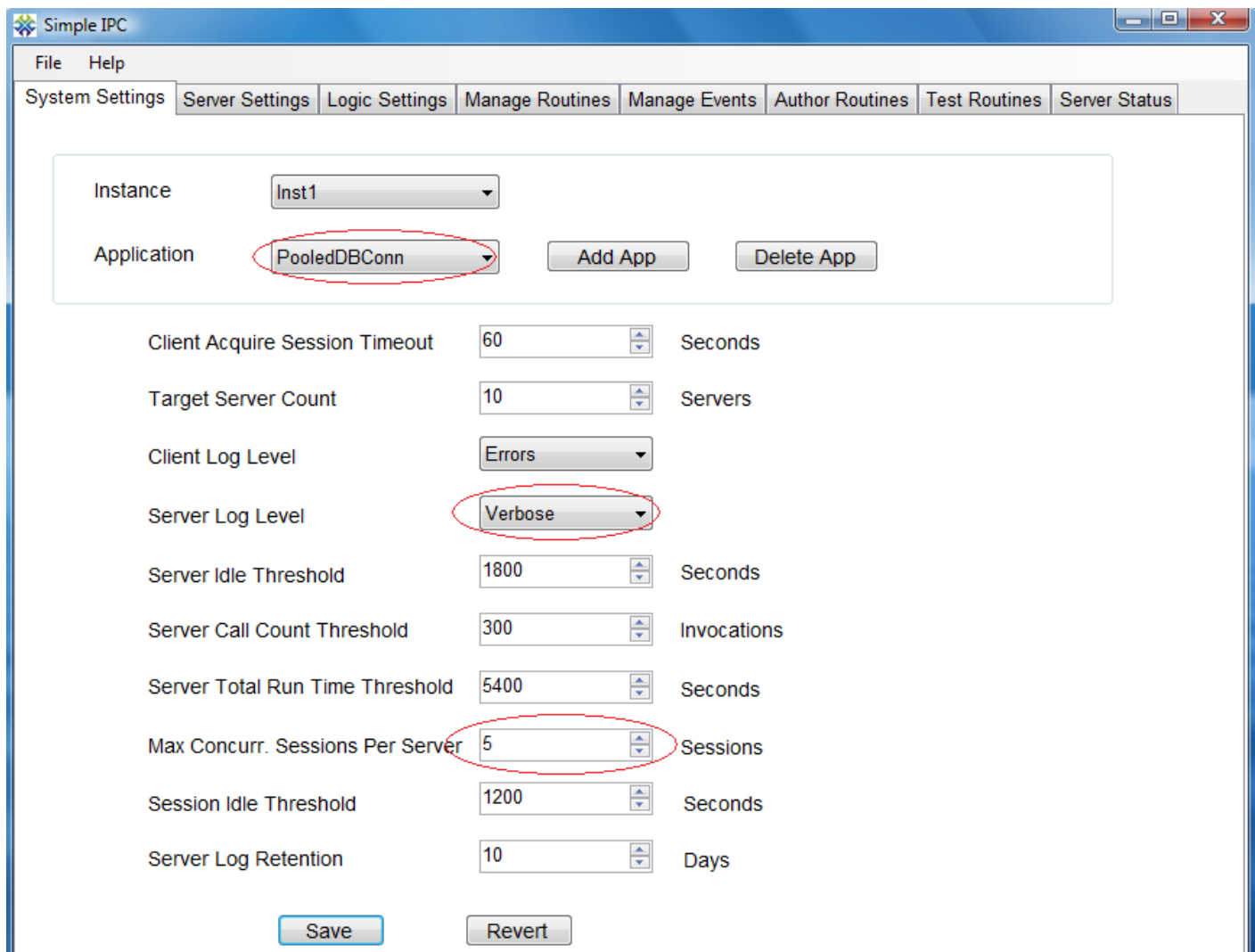
Copyright © 2009 Cognitier, Inc. All rights reserved.

Cognitier and the Cognitier logo are trademarks of Cognitier, Inc. All other company and product names referred to may be trademarks of their respective owners. This documentation is copyrighted material and is intended exclusively for use by licensed users of Cognitier software. The information in this document is subject to change without notice.

Exercise 3 – Object Pooling

The point of this exercise is to demonstrate how to create a number of object instances when the IPC server starts up, and to re-use these objects as various clients obtain and release sessions throughout the life of the IPC server. In this case, we will once again be dealing with database connections. We want to create a server start-up routine that obtains and opens several database connections and puts them into the server context. Within the routine that will be invoked by name by external clients, we will get a database connection out of the named object pool and use it to retrieve information from the database. In the server exit routine, we will get all of the connections from the pool and close them.

Step 1: Create a new application for which our server start-up and shut-down routines will fire. Go to the System Settings tab and create a new application and call it PooledDBConn. Set the server log level for this application to Verbose. Set the maximum number of concurrent sessions to five.



Step 2: Server Settings and Logic Settings are specific to an Instance/Application combination. We need a connection string for our database connections. In this case, we will probably use the same connection string, but we must create a new Server Setting for our new application. Go to the Server Settings tab and toggle the Instance selection in order to refresh the contents of the Applications drop-down. Select the PooledDBConn application and add a new Server Setting called “MyDBConnStr” (i.e. we can give the Server Setting the same name we used before since this setting is for a different application). Enter a valid value for the connection string (i.e. probably the same one we used earlier) and save the setting.

Step 3: Create the server start-up routine. Go to the Author Routines tab of the CTConsole. Create a new Source Code file. Call the file PooledObjSrvrStart, and select the BasicServerOp1 template. Add a reference to C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.Data.dll. For this exercise, use the following code to create and open the database connections and put them into a named object pool.

```
import System;
import System.Collections;
```

```

import System.Reflection;
import System.Data.SqlClient;

[assembly:AssemblyVersion("1.0.0.0")]

public class PooledObjSrvrStart implements CTSHost.IBasicServerOp1
{
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor) : System.Boolean
    {
        //Give the object pool a name. Remember to use the same name in the other
routines
        var POOLNAME: System.String = "ConnectionsPool";
        var oLogUtil: CTCommon.LogUtil;
        oLogUtil = CTCommon.LogUtil.Instance;

        var oConn: SqlConnection;
        var bRet: System.Boolean;

        oLogUtil = CTCommon.LogUtil.Instance;
        bRet = false;
        try
        {
            //Get the database connection string Server Setting
            var sConnStr: System.String = String.Empty;

            if (oServerContextAccessor.GetServerSetting("MyDBConnStr") != null)
            {
                sConnStr = oServerContextAccessor.GetServerSetting("MyDBConnStr");
            }
            if (sConnStr.length == 0)
            {
                throw new Exception("Unable to obtain connection string");
            }

            //Get the setting value for the max number of concurrent sessions for the
server
            var iTargetClientCount: int =
oServerContextAccessor.ServerConcurrentClientLimit;

```

```

    //Within a loop, create and open one database connection per allowed
concurrent session.
    //Put the connections into an ArrayList
    var iObjectCount: int = 0;
    var oPooledConnectionsAL: ArrayList = new ArrayList();
    while (iObjectCount < iTargetClientCount)
    {
        oConn = new SqlConnection();
        oConn.ConnectionString = sConnStr;
        oConn.Open();
        oPooledConnectionsAL.Add(oConn);
        iObjectCount++;
    }

    //Set the collection of connections into an object pool
    if (oServerContextAccessor.SetPooledObjects(PPOOLNAME,
oPooledConnectionsAL) != 1)
    {
        throw new Exception("Unable to establish object pool " + PPOOLNAME);
    }

    bRet = true;
}
catch (e)
{
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;
} //end function
} //end class

```

Step 4: Register the routine we just created. Go to the Manage Routines tab and click the "Refresh List" button so that our new dll will appear in the dlls drop-down. Select PooledObjSrvrStart.dll, enter a routine name of PooledObjSrvrStart, and save it.

Step 5: Associate the routine we just created with the "On Server Start" event of the PooledDBConn application. Go to the Manage Events tab and toggle the Instance selection in order to refresh the list in the Applications drop-down. Select the PooledDBConn application. Select the PooledObjSrvrStart routine from the drop-down for the "On Server Start" event and click the Save button.

Step 6: Create the routine to be invoked by name by external clients. Go to the Author Routines tab of the CTConsole. Create a new Source Code file. Call the file PooledObjBasicCall, and select the BasicRoutineRun1 template. Add a reference to C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.Data.dll. For this exercise, use the following code to get the database connection out of the object pool, query the database for a value, and put the value into the output arguments collection.

```
import System;
import System.Collections;
import System.Reflection;
import System.Data.SqlClient;

[assembly:AssemblyVersion("1.0.0.0")]

public class PooledObjBasicCall implements CTSHost.IBasicRoutineRun1
{
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor, oSessionContextAccessor:
CTSHost.SessionContextAccessor, oCallParamsAccessor:
CTSHost.CallParamsAccessor) : System.Boolean
    {
        //Be sure to use the same name for the object pool that we used when we
created it
        var POOLNAME: System.String = "ConnectionsPool";
        var oLogUtil: CTCommon.LogUtil;
        oLogUtil = CTCommon.LogUtil.Instance;

        var oConn: SqlConnection;
        var oCmd: SqlCommand;
        var oReader: SqlDataReader;
        var bRet: System.Boolean;

        oLogUtil = CTCommon.LogUtil.Instance;
        bRet = false;
        try
        {
            //Get the connection that is allocated for our session out of the object
pool.
            //We have to convert the object to the proper data type.
            //Make note of the calls to oSessionContextAccessor.SessionID to get the
current session id
```

```

        //and oServerContextAccessor.GetPooledObject to get the object from the
named pool for our session
        oConn =
System.Convert.ChangeType(oServerContextAccessor.GetPooledObject(POOLNAME,
oSessionContextAccessor.SessionID), SqlConnection);
        oCmd = new SqlCommand;
        oCmd.Connection = oConn;
        oCmd.CommandText = "SELECT TOP 1 email_addr FROM Employee";
        oReader = oCmd.ExecuteReader();
        if (oReader.Read())
        {
oCallParamsAccessor.OutputArgs.Add(System.Convert.ToString(oReader(0)));
        }
        oReader.Close();

        bRet = true;

    }
catch (e)
{
    //Set errors into errors collection - must be strings
    oCallParamsAccessor.OutputErrors.Add(e.Message);

    //Also write errors to the server log file
    //Log levels are Errors (1), Warnings (2), Info (3), Verbose (4)
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;

} //end function

} //end class

```

Step 7: Register the routine we just created. Just as before, go to the Manage Routines tab and click the "Refresh List" button so that our new dll will appear in the dlls drop-down. Select PooledObjBasicCall.dll, enter a routine name of PooledObjBasicCall, and save it.

Step 8: Create the server exit routine. Go to the Author Routines tab of the CTConsole. Create a new Source Code file. Call the file PooledObjSvrEnd, and select the BasicServerOp1 template. Add a reference to

C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.Data.dll. For this exercise, use the following code to get the database connections out of the object pool and close them.

```
import System;
import System.Collections;
import System.Reflection;
import System.Data.SqlClient;
import System.Data;

[assembly: AssemblyVersion("1.0.0.0")]

public class PooledObjSrvrEnd implements CTSHost.IBasicServerOp1
{
    public function RunRoutine(oServerContextAccessor:
CTSHost.ServerContextAccessor) : System.Boolean
    {
        //Be sure to use the same name for the object pool that we used when we
created it
        var POOLNAME: System.String = "ConnectionsPool";
        var oLogUtil: CTCommon.LogUtil;
        oLogUtil = CTCommon.LogUtil.Instance;

        var oConn: SqlConnection;
        var bRet: System.Boolean;

        oLogUtil = CTCommon.LogUtil.Instance;
        bRet = false;
        try
        {
            //Call oServerContextAccessor.GetAllObjectsFromPool to get the ArrayList
of connections
            var oPooledConnectionsAL: ArrayList =
oServerContextAccessor.GetAllObjectsFromPool(POOLNAME);
            var iPooledObjSum: int = oPooledConnectionsAL.Count;
            var iObjectCount: int = 0;
            //Loop through the elements in the ArrayList, convert them to the
SqlConnection datatype, and close them.
            //There is no need to remove the objects themselves from the pool because
the process is about to exit.
            while (iObjectCount < iPooledObjSum)
            {
```

```

        oConn = System.Convert.ChangeType(oPooledConnectionsAL(iObjectCount),
SqlConnection);
        if (oConn.State == ConnectionState.Open)
        {
            oConn.Close();
        }
        iObjectCount++;
    }

    bRet = true;
}
catch (e)
{
    //Write errors to the server log file
    oLogUtil.LogOutput(CTCommon.Constants.LogLevel.Errors,
CTCommon.Constants.LogOpType.RoutineExecution, e);
}
return bRet;

} //end function
} //end class

```

Step 9: Register the routine we just created. Go to the Manage Routines tab and click the "Refresh List" button so that our new dll will appear in the dlls drop-down. Select PooledObjSvrEnd.dll, enter a routine name of PooledObjSvrEnd, and save it.

Step 10: Associate the routine we just created with the "On Server End" event of the PooledDBConn application. Go to the Manage Events tab and select the App1 application and then the PooledDBConn application (in order to refresh the lists of routines in the event drop-downs). Select the PooledDBConn application. Select the PooledObjSvrEnd routine from the drop-down for the "On Server End" event and click the Save button.

Step 11: Test your work. Go to the Test Routines tab of the CTConsole. Toggle the selected Instance from "Inst1" to "Inst2" and back to "Inst1" in order to refresh the list of applications. Select the "PooledDBConn" application. Click the refresh icon to the right of the Routine drop-down to refresh the contents of the drop-down. Select the "PooledObjBasicCall". It does not matter which Entry Point we use, but leave the "Out Of Proc COM" entry point selected. Likewise, it does not matter which Test Case is selected. Leave the default selections of one thread/session and one call per session and click the "Start" button. Observe the output in the text area at the bottom of the form and make sure it matches your expectations. You may also increase the number of sessions and number of calls per session and monitor the database itself to ensure that the number of connections does not increase beyond expectations while the test is running.

