

Installer – Custom Textboxes in UI problem

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-02/0508.html>

From: Craig (*cscheets_at_remoovdis.kc.rr.com*)

Date: 02/05/04

Date: Thu, 5 Feb 2004 13:44:26 -0600

I have added a 'Textboxes (A)' to my UI installer project along with a custom action to pass the value back to a class I've written to override the void Install function. As long as the text is very simple everything is working just fine. I've tested it by writing the string out to a text file to test it.

However, the trouble comes in when I use the textbox for what I really intended, a ADO.Net Connection String. The textbox is to allow the user to put the connection string and I intend to test the db connectivity and write it to the registry during installation. However, when I enter a real connection string in the box, it chokes and gives an error — presumably because of either the semicolon, space, or equals characters in it. Is there a list of characters that can't be input? Do I need to escape them somehow before passing them?

(notice the beginning of the exception message is `_PART_` of my connection string, it's trying to parse that string into multiple context parameters I think). I get the following error

Exception occurred while initializing the installation:
System.ArgumentException: File Source=workstation;Initial does not exist.
If this parameter is used as an installer option, the format must be /key=[value]..
(OK Button)

Again, everything works when the string is simple, so I'm sure I'm passing it OK. Just to provide all the info – I've named the Edit1Property to 'CONNSTRING' and the custom action 'CustomActionData' is /connString=[CONNSTRING]. The exact string I input in the box and overridden install function is below:

String Input:

```
Data Source=workstation;Initial Catalog=serverPerformance;User  
ID=serverPerformance;Password=server;
```

My install function:

```
[RunInstaller(true)]
```

```
public class ProjectInstaller : System.Configuration.Install.Installer
{
public override void Install(IDictionary state)
{
string connString = this.Context.Parameters["connString"];

if(connString == "")
{
throw new InstallException("The database conection information is not
optional. Please obtain this information from your system administrator and
run setup again.");
}

System.Data.SqlClient.SqlConnection myConn = new SqlConnection();

try
{
string queryCount = "SELECT count(*) FROM sysobjects WHERE name like
'sample%' AND xtype = 'u'";

myConn = new System.Data.SqlClient.SqlConnection(connString);

System.Data.SqlClient.SqlCommand cmdCount = new
System.Data.SqlClient.SqlCommand(queryCount,myConn);

myConn.Open();

int tableCount = (int) cmdCount.ExecuteScalar();

if (tableCount<2)
{
throw new InstallException("The connection to the database was made
successfully, but the correct database structure doesn't exist. Please use a
different database or refer to the documentation to build the database
structure and then reinstall this service.");
}
}
}
```

```
catch (System.ArgumentException argEx)

{

throw new InstallException("Error validating provided SQL connection string.
Please check your connection string and run setup again.\r\n\r\nError: "
+argEx.Message);

}

catch (System.Data.SqlClient.SqlException sqlEx)

{

throw new InstallException("Error connecting to database and executing a
query to test the database availability and structure. Please check your
connection string and and database server and run setup again.\r\n\r\nError:
" +sqlEx.Message);

}

catch (Exception ex)

{

throw new InstallException("There was an error attempting to validate your
connection string and database structure. Please check your connection
string and database server and run setup again.\r\n\r\nError: "
+ex.Message);

}

finally

{

if (myConn.State != ConnectionState.Closed)

myConn.Close();

}

}

}
```