

Description: Guide to generating BridgeWorX Custom Tasks.
OS Requirement: Win XP/Server 2003/ Server 2008/Vista/Windows 7
General Requirement: Knowledge and installation of Microsoft Visual Studio 2005 or Visual Studio 2008 and the .NET Framework 2.0.

Introduction

BridgeWorX has the concept of a Custom component to provide a customized data source to be used within the BridgeWorX Framework. The BizViz Custom Task Component is used to provide this functionality.

NOTE: Although we support Microsoft Visual Studio 2005 and Visual Studio 2008, the example in this document will be using Visual Studio 2005

Installing the BizViz Custom Task Toolkit

The installation steps are as follows:

1. Locate your BizViz V9.X disc.
2. Navigate to the Tools\BizViz Custom Task Wizards folder on the BizViz V9.X disc.
3. Copy the zip file *BizVizCustomTask.zip* to My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#.
4. Use the Microsoft .NET Framework 2.0 Configuration utility from the Administrative Tools to add the *BizVizCustomTaskWizard.dll* to the Global Assembly Cache (GAC) from the same BizViz disc folder location as shown in **Figure 1**.

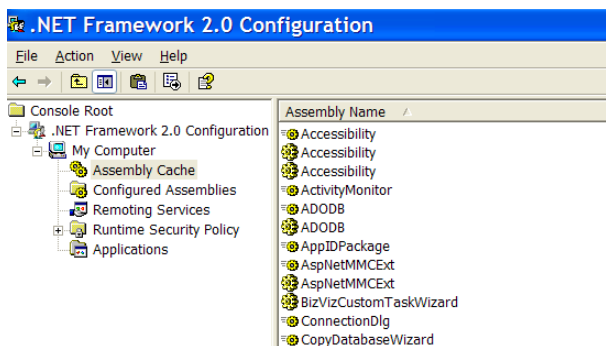


Figure 1 - Adding BizViz Custom Task Wizard to Global Assembly Cache

Migrating Existing BridgeWorX Custom Task Components

As of the version 9.1 release, BridgeWorX is now based on .NET Framework 2.0. This requires that all existing BridgeWorX Custom Tasks be upgraded to the same level, by

re-compilation in Visual Studio 2005. Existing BridgeWorX Custom Task components from version 8 can be migrated to BizViz Custom Task Version 9.X as follows:

1. Install the .NET Framework 2.0 and Visual Studio 2005.
2. Load your existing Visual Studio 2003 BridgeWorX Custom Task project into Visual Studio 2005.
3. A Visual Studio 2005 dialog will prompt you to upgrade your existing project. Accept the option to upgrade your existing project and Visual Studio 2005 will upgrade your existing solution.
4. Remove the existing BridgeWorX version 8 *BwxCustomTaskBridge.dll* reference from your project.
5. Navigate to ..\BizViz\BridgeWorX\BwxServices and add the V9.X *BwxCustomTaskBridge.dll* reference to your project as shown in Figure 2.
6. Recompile and take care of any compilation errors.
7. Your updated Custom Task is ready to be used. If it is deployed on a different folder than the project build folder, make sure that the updated BridgeWorX version 9.X *BwxCustomTaskBridge.dll* is also copied to the deployment folder.

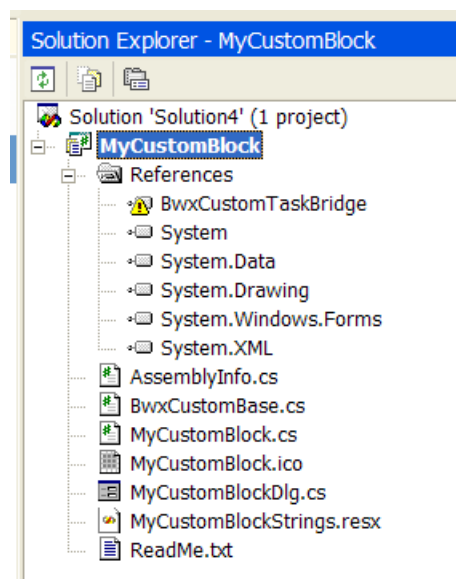


Figure 2 - Updating BwxCustomTaskBridge.dll

BizViz Custom Blocks and Assemblies

Custom blocks are configured in the BridgeWorX Configurator require the assistance of external libraries (.NET assemblies contained in .dll files) in order to perform their Custom Tasks. These libraries contain resource information, such as icons and strings that the BridgeWorX Configurator uses to represent the Custom Task in its configuration dialogs. More importantly, the libraries contain custom C# managed code that the BridgeWorX runtime engine uses to perform the task at hand.

Simplifying the Custom Code

The task of writing custom code, particularly the task of writing code that will drive a component of a closed-source application, is often regarded as a daunting and formidable task. To that end, BridgeWorX has gone to great lengths to simplify the code-writing task of the end user by supplying a common interface to which all Custom modules conform. BridgeWorX supplies a toolkit designed for the Microsoft Visual Studio .NET 2005 development environment that creates a compiler-ready project with minimal series of steps. This project implements the required common interface for you and supplies a rich set of comments. These comments, along with this document, point you to the specific regions of code that you are to modify to give your Custom component its desired functionality.

Running the Custom Component Wizard

The BizViz Custom Component Toolkit takes the form of a Microsoft Visual Studio 2005 Project Wizard in C# and therefore requires that Microsoft Visual Studio 2005 be installed on your PC.

1. Launch Visual Studio 2005
2. Select “New Project” at the “Start” page or select the File → New → Project menu from the menu bar.
3. In the “New Project” dialog that appears, select the “Visual C# Projects” folder in the “Project Types” tree.
4. If you have the BizViz Custom Task Component template installed, the dialog should look like **Figure 3**.

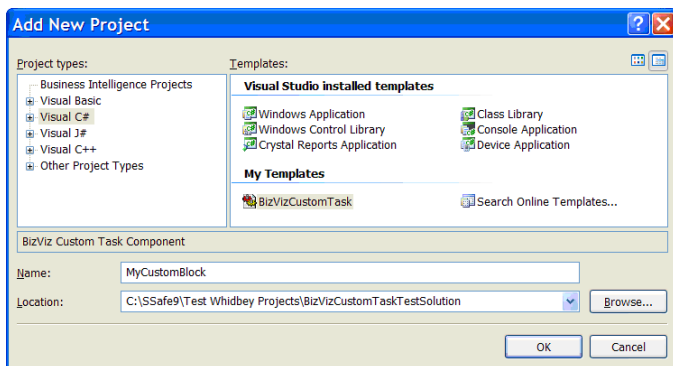


Figure 3 - Selecting the BizViz Custom Component Toolkit

5. Notice the “BizVizCustomTask” entry among the templates to the right.
6. Select this item and supply a name for your Custom Task. This name will become the name of your project, your Custom code implementation class, and your resulting .dll file after compilation.
7. Upon pressing the “OK” button, the BizViz Custom Component Wizard begins as shown in Figure 4.

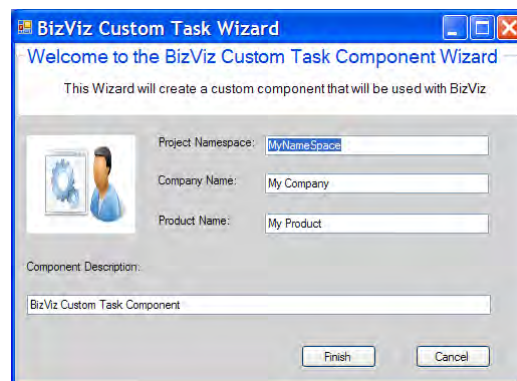


Figure 4 - Custom Component Wizard

8. This first and only page of the wizard allows you to supply some information about your Custom component.
9. The “Project Namespace” field is for your use and does not affect interaction with the BridgeWorX runtime. The value you supply will become the default namespace for the created project and will be written as the namespace for your main coding class. It is suggested to avoid spaces and characters like “\$”, “*”, etc. in the Namespace field. Ideally, something like CompanyName.Product.Class may be a good starting point.
10. The “Company Name” and “Product Name” and “Component Description” fields help users identify your Custom module in the BridgeWorX Diagram Designer. The strings will also be written into the assembly information of the library itself and will become properties of the compiled .dll file.

Exploring the New Project

Upon pressing the “Finish” button in the wizard, a new project is created that implements your Custom component. The Solution Explorer shows a list of all of the files contained in the newly created project. If you do not see the Solution Explorer, go to the View → Solution Explorer menu as shown in Figure 5.

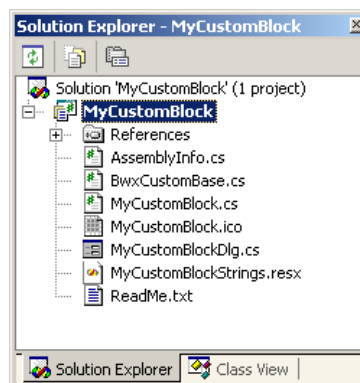


Figure 5 - New Project's Solution Explorer

In the Solution Explorer, you will notice the presence of an icon, a ReadMe.txt, a resource file, an assembly information file and a few source files. No additional files are necessary for a working Custom component. Notice that three of the source files begin with the “MyCustomBlock” string supplied at the “New Project” dialog.

In the Solution explorer,

1. Expand the References branch as shown in **Figure 6**.
2. Remove the BwxCustomTaskBridge reference.
3. Right mouse click to add a new reference to the *BwxCustomTaskBridge.dll*, located under the “BizViz\BridgeWorX\BwxServices” folder.

NOTE: Keep in mind that this .NET assembly needs to be physically located in the same folder where your Custom Task will compile and run. For convenience purposes, the BridgeWorX installation will create a CustomAssemblies folder under the BridgeWorX installation directory. If the user decides to use this folder as the target folder for compiling and executing the Custom Task assembly, then a copy of the *BwxCustomTaskBridge.dll* should be placed there as well.

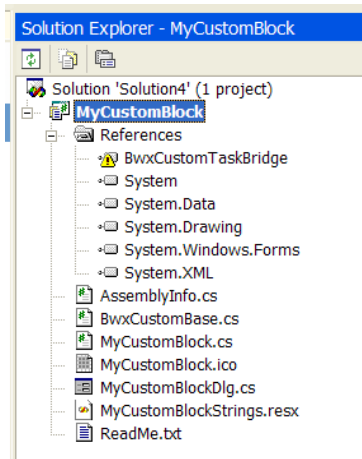


Figure 6 - Updating the references for the Custom Task

Project Files

AssemblyInfo.cs

This file is a standard .NET assembly information file. You will notice that the company name, product name, and description supplied in the wizard appear as fields in this file. Modify these fields, along with the version, however you see fit.

```
// General Information about an assembly is
// controlled through the following set of
// attributes. Change these attribute values to
// modify the information associated with an
// assembly.
[assembly: AssemblyTitle("BizViz Custom Task
Component")]
[assembly: AssemblyDescription("BizViz Custom Task
Component Created with BizViz Custom toolkit")]
[assembly: AssemblyConfiguration("")]
```

```
[assembly: AssemblyCompany("My Company")]
[assembly: AssemblyProduct("My Product")]
[assembly: AssemblyCopyright("Copyright© 2010.")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

(ProjectName).ico

This file supplies an icon to the Custom component that the BridgeWorX configurator will use as a graphical representation of the component. Opening this file will launch the icon editor in Visual Studio. Customize this icon in any way you see fit.

(ProjectName)Strings.resx

This file holds any strings that your Custom component may reference. By default, the Product Name and Company Name supplied in the wizard appear as string resources as shown in Figure 7.

Data:					
Data for data					
	name	value	comment	type	mimetype
▶	Product Name	Custom Task Test	Library Info	(null)	(null)
	Company Name	ICONICS	Library Info	(null)	(null)
*					

Figure 7 - New Project's Resource File

NOTE: DO NOT DELETE OR MODIFY THE “NAME” FIELD OF THESE ENTRIES as they are used by the main code file to supply product and company information to the BridgeWorX Designer. You can customize the “value” field at any time.

ReadMe.txt

This file supplies information about your newly created project and gives a brief description of the files contained therein.

BwxCustomBase.cs

This file supplies compatibility with the common interface for all Custom components and **MUST NOT** be modified in any way. The interface that this file implements is encapsulated in the BwxCustomTaskBridge assembly seen in the project's references. The main code file inherits this class. All customizations should be done through the main code file.

(ProjectName)Dlg.cs

This file implements a sample dialog, viewable from the BridgeWorX configurator as shown in Figure 8.

This dialog is the mechanism by which the end user can persist data that is global to all Custom method calls. The value of the “Sample Text” field is persisted in the BridgeWorX database upon pressing “OK” in the dialog. The end user can add his or her own custom parameters by simply following the model by which “Sample Text” is implemented in the underlying code.

Because of the nature of the Custom Task, the BridgeWorX Diagram Designer or runtime tools can only perform validation on the input and output data mappings. It is strongly recommended that the user validates the Custom Task configuration data prior to saving the configuration information within the BridgeWorX database.

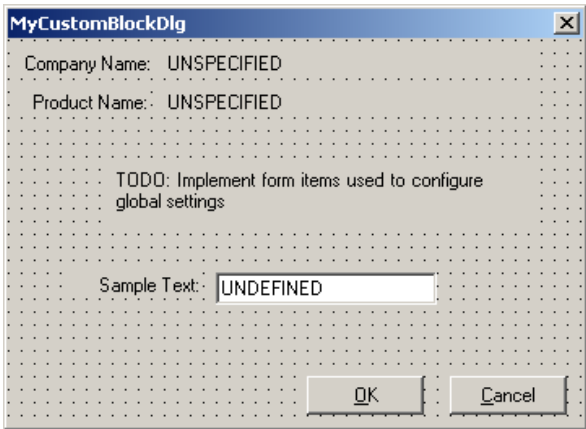


Figure 8 - Default configuration dialog

NOTE: The BizViz Custom Task uses a DataSet as the storage media for its configuration data. Once the configuration dialog has been completed, the DataSet which holds the configuration data will be saved inside the BridgeWorX database with the user-entered information and it will be used to initialize the Custom Task during runtime execution

(ProjectName).cs

This is the main code file of the project. All code written for Custom Tasks is written to this file.

NOTE: Several of the methods in this file are surrounded by “region” flags and are, by default, “closed” when viewed in the code editor. These methods stay the way they are and should not be edited.

The “CustomMethod” function deserves the most attention. This function shows an example of how one would implement a Custom Task. The function shows how to take a string input and “echo” it back in a DataSet. DataSet objects are convenient for sending multiple values back to BridgeWorX. You can rename “CustomMethod” to any legal C# function name.

The number and type of parameters can vary as long as the parameter types and return values are one of those in **Error! eference source not found..**

Table 1 - Return Types for Custom Methods

bool	byte	char	decimal	double	short
int	long	sbyte	float	ushort	uint
ulong	string	bool[]	byte[]	char[]	decimal[]
double[]	short[]	int[]	long[]	sbyte[]	float[]
ushort[]	uint[]	ulong[]	string[]	DataSet	

You are not restricted to only one Custom method. You can add as many Custom methods as you wish to this class. Anything declared “public” is accessible by BridgeWorX, as long as it contains simple data types for parameters.

It is required that each Custom data source method sets the *TransactionStatus m_LastTransactionStatus* properly, so that the BridgeWorX runtime engine will detect if the method succeeded or failed. Utilizing try/catch exception mechanisms is a good programming practice to handle any code errors within the Custom Task itself and then update the BridgeWorX runtime engine with the corresponding TransactionStatus value. In addition, the Custom data source developer can use the *m_szLastError* member variable to pass error information to the BridgeWorX runtime engine. The *TransactionStatus* can take the following set of values:

```
//enumerated type for all transaction status codes
public enum TransactionStatus
{
    StatusComplete = 0,
    StatusFail,
    StatusSucceed
}
```

- **StatusFail** should be used when the Custom data source method failed or serious errors occurred
- **StatusSucceed** should be used when the Custom data source method completed successfully without any errors.
- **StatusComplete** can be used in the case where the Custom Task method completed with some errors, which the end user does not consider serious enough to fail the Custom data source method call.

It is suggested that the return of the Custom Task method is formulated as a DataSet. In this way, the return can be mapped in a more effective and flexible way within the BridgeWorX configuration tools.

Completing the Custom Module

Since the Custom project is designed to compile “out of the box”, you can compile the module as-is, without writing a single line of code and it will operate with default behavior in BridgeWorX. To give the module custom behavior, modify the code, strings and icon as described in the “Project Files” section.

Once your Custom code has been added, compile the project using the “Build → Build Solution” menu option. If the Custom code is error free, a Custom (project name).dll file is created. In BridgeWorX configurator, under the Custom block configuration, you can now browse for this newly created (project name).dll file and select the appropriate Custom function.

NOTE: For the current version of BridgeWorX, compiling a new version of the Custom Task assembly requires that it is not loaded in process memory by the BridgeWorX configurator or the BridgeWorX service. Stopping the BridgeWorX runtime service ensures this.

New Advanced Support for BizViz Custom Tasks in BridgeWorX

The BizViz Custom Task Component in BridgeWorX has been greatly enhanced to support advanced functionality in the areas of array support, DataSet input mapping, and DataSet output mapping to a database destination.

New user-friendly array mapping has been introduced for array and DataSet inputs to the Custom Task. For example, in the case of database inputs to a Custom Task array input parameter, the user can have options to map the entire return or only a certain number of records returned from the database. In addition, expressions can be applied as well prior to sending the data to the Custom Task.

DataSet input support for Custom Tasks have also been greatly enhanced for both OPC DA and database sources to the Custom Task. Users now have the ability to shape a DataSet input to a Custom Task by defining tables and column mappings and applying expressions to them.

NOTE: In a customized DataSet input parameter, the user should access the internal tables by index rather than by name. Although the BridgeWorX Configurator allows the user to name the tables, the working approach is to access each DataTable within a DataSet input to the Custom Task Component by DataTable index.

In the case of using OPC DA as a DataSet input parameter for a Custom Task, the user can :

- Shape the DataSet inputs
- Apply specific attributes to expressions to get a tabular OPC DA input, which includes OPC DA tag values, qualities as well as time stamps in UTC format. The user can use expressions like “x= {{#Quality#}}” or “x= {{#TimeStamp#}}”, to get a tabular input structure for OPC DA data as shown in **Figure 9**.

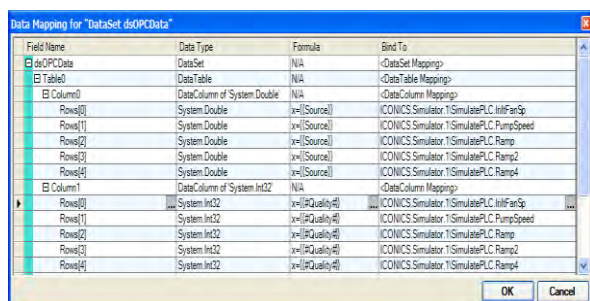


Figure 9 - Custom Task DataSet input shaping for OPC DA tags

Finally, the Custom Task DataSet output support has been enhanced to allow for multiple database updates, inserts or stored procedure calls. In addition, expressions can be applied as well. Some of the new options allow for applying transaction scope, which ensures that all records are updated in the database or the entire operation is rolled back, appending to an existing table or truncating it first and then inserting the new records etc as shown in **Figure 10**.

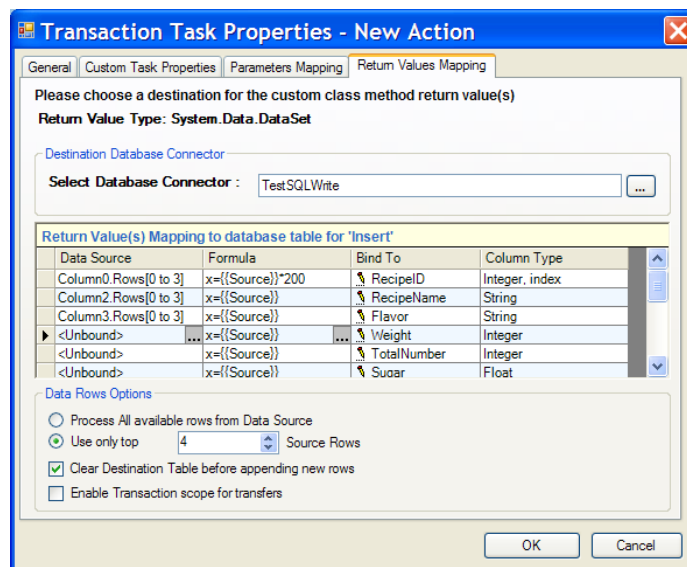


Figure 10 - Custom Task DataSet output mapping and related options

NOTE: If BridgeWorX Global variables are used as inputs to the Custom Task Component, they will be transferred to the Custom Task inputs as one of System.Int32, System.Double, System.String or System.DateTime data types. If the Custom Task is expecting a different data type, it may be necessary to type cast the received Global variable data type to the desired one.