SBW Visual Studio 2003[1] Add-in

Frank Bergmann,

Version 1.0, 2005-09-09

This document describes the SBW Visual Studio 2003 Add in. This Add in allows creating Wrappers around SBW Modules in one of the following three languages:

- C#
- VB.NET
- C++

This makes it very easy to access SBW modules and helps o achieve SBWs main goal: to enable different software applications to easily interact with each other.

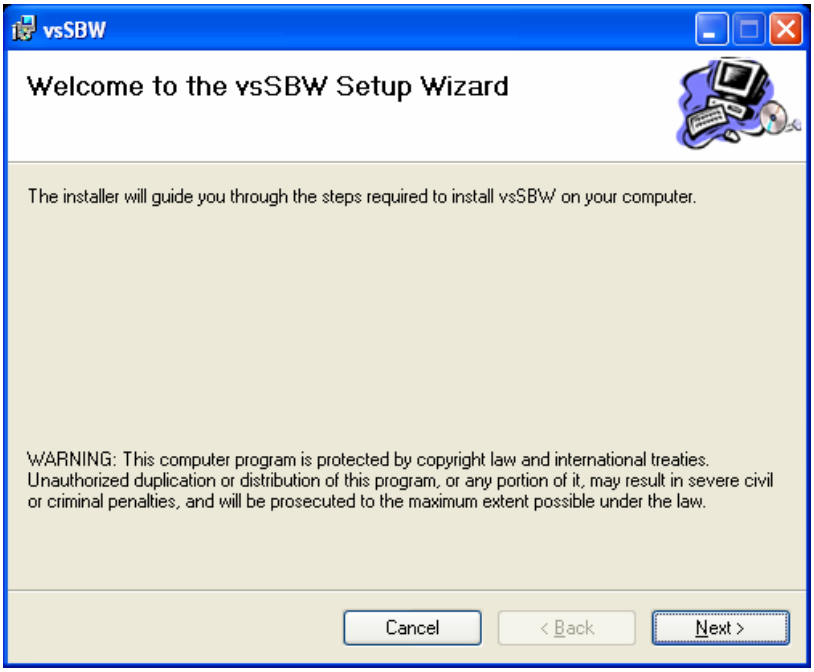
[1] Microsoft Development Environment 2003, Copyright © 1987-2002 Microsoft Corporation

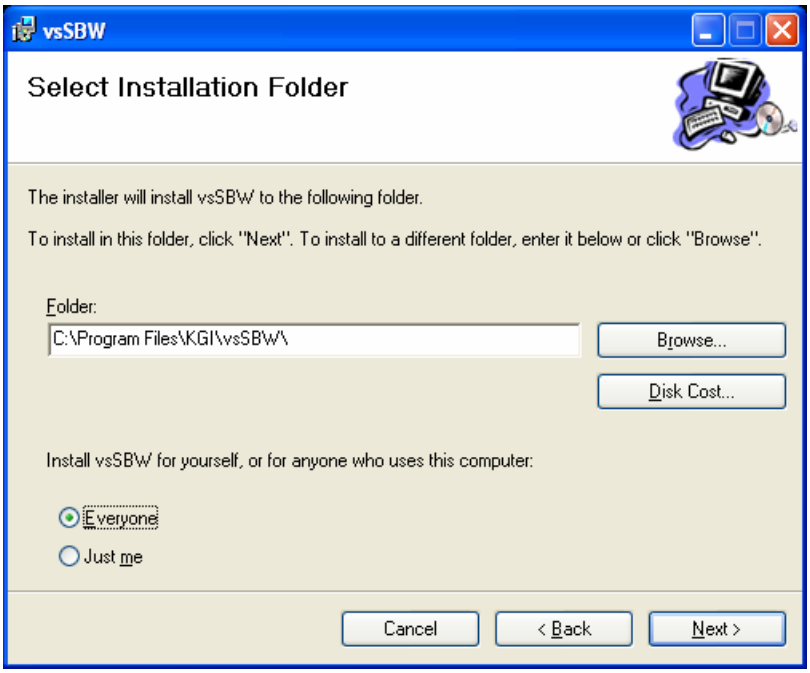
Installing the Add in

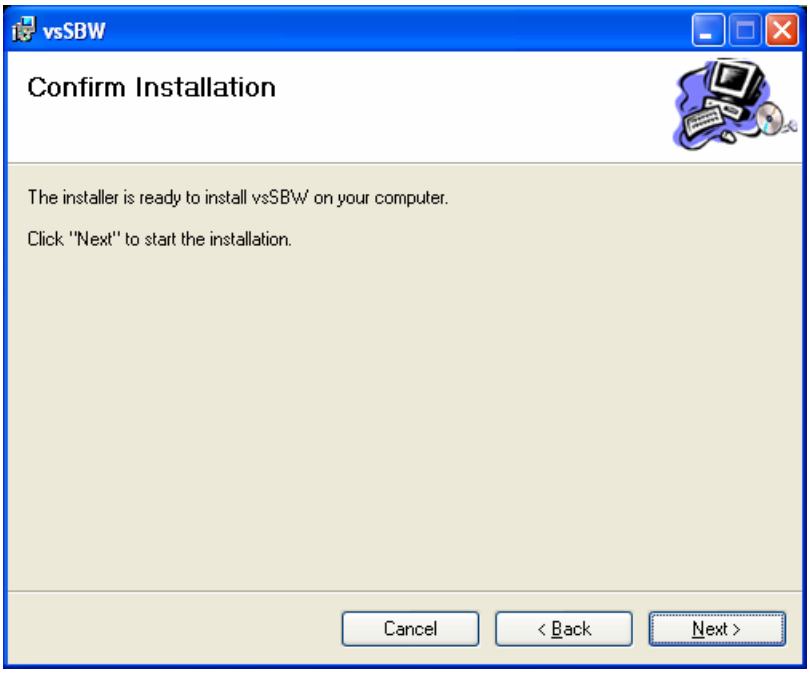
The installation process is pretty straightforward:

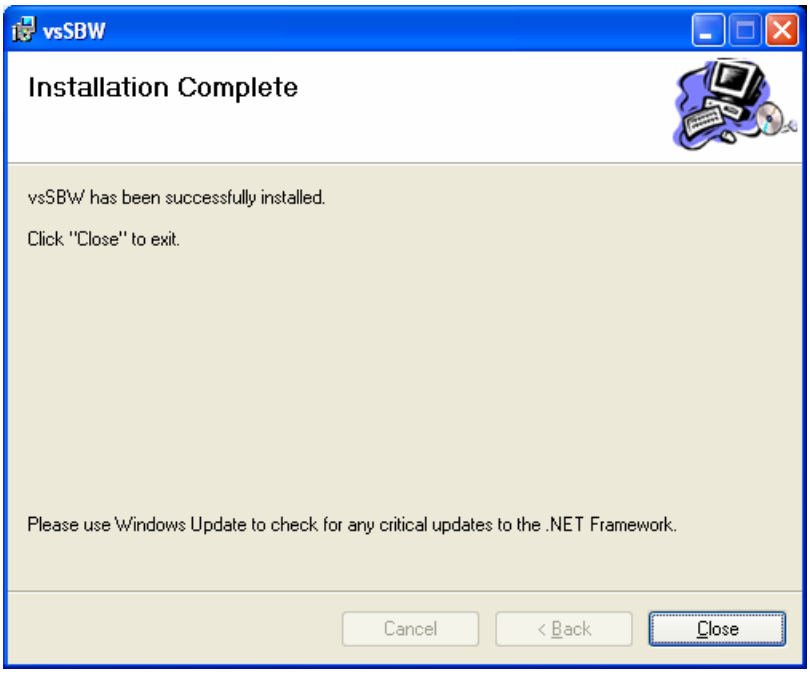
- Download the most recent version of SBW (at least 2.4.0) from: http://www.sys-bio.org
- download the Add in: from: http://public.kgi.edu/~fbergman/SBW.html
- close open instances of Visual Studio 2003
- (uninstall previous versions of the plugin)
- double click the **vsSBWSetup.msi**

The following screens will appear:




vsSBW
Select Installation Folder
The installer will install vsSBW to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\KGI\vsSBW\
Browse...
Disk Cost...
Install vsSBW for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >


vsSBW
Confirm Installation
The installer is ready to install vsSBW on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >



Simply follow the instructions on the screen, pressing “Next” and “Close” will work in most cases.

Next start Visual Studio 2003 to verify that the installation process was successful. Once Visual Studio was started up, click on the Tool menu. There you should see an entry called “SBW Integration”.

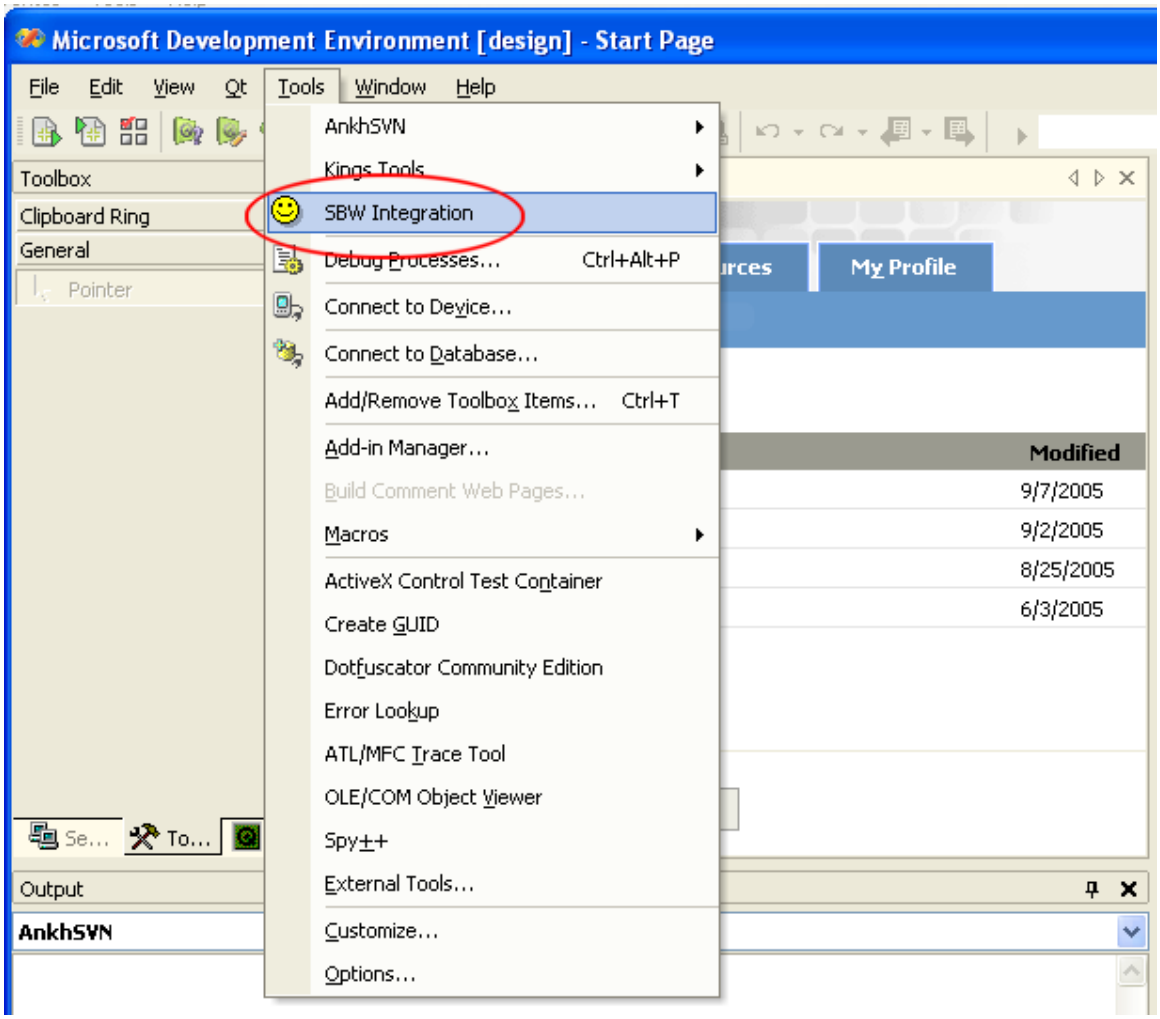


Figure 1: successful installation of the SBW Integration

Should the entry is not visible see the object, the Wrapper might stop working with future versions of the module.

Upon a click on “Create” the wrapper code will be created and a new File will be added to your solution. Once that is done we can simply perform a simulation run like this:

The wrapper implements every call to the SBW module as public static (public shared for VB) method. Hence no Object has to be created to call a module. (This reflects the stateless SBW API, if a Module requires a state it will either implement this behavior via different parameters or via a protocol that defines the order in which methods can be called.)

The last call in “Figure 6: C# simulating a SBML model” represents the disconnection from SBW. This should be done at the end of the program. If it is omitted the application will not terminate as it listens for possible incoming calls.

Known Limitations / Troubleshooting section below.

Using the Add in

Once the Systems Biology Workbench along with the Add In is installed one can use it to access SBW Modules. This section will demonstrate how to create a simple C# Console Application that allows simulating a SBML model given as command line argument.

First create a new C# Console Application Project (selecting File\New Project):

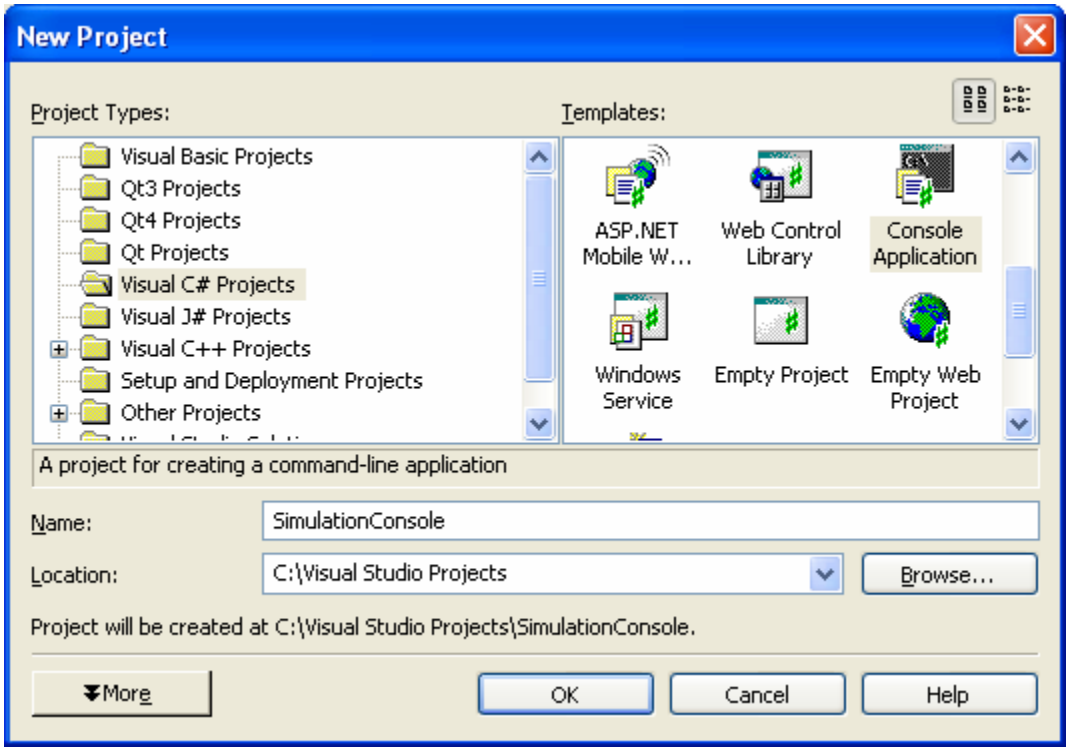


Figure 2: Creating the example project

Note: It would also have been possible to select a Visual Basic Project or a Visual C++ Project.

The first thing to do in this example would be to parse the command line argument. In this simple case let use assume that the user will invoke the application with:

SimulationConsole.exe –m | --model <sbml file>

Meaning he gives either the parameter "-m" or "—model" followed by the filename of the SBML file to simulate. This could be done in the following way:

```
#region /// reading the SBML
       // parse the command line argument
       string sFilename = null;
       for (int i = 0; i < args.Length; i++)
       {
              if ((args[i].ToLower() == "-m" ||
                     args[i].ToLower() == "--model")
                     && i+1 < args.Length)
              {
                     sFilename = args[i+1];
                     break;
              }
       }
       // if we don't have a model yet, bail out.
       if (sFilename == null)
       {
              Console.WriteLine(
                     "usage: SimulationConsole.exe –m | --model <sbml file>");
              Environment.Exit(-1);
       }

       // now we have a filename ... try to get the sbml.
       string sSBML = null;

       try
       {
              StreamReader oReader = new StreamReader(sFilename);
              sSBML = oReader.ReadToEnd();
              oReader.Close();
       }
       catch (FileNotFoundException e)
       {
              Console.WriteLine("The file : " + sFilename +
                     " could not be found (" + e.Message + ")");
              Environment.Exit(-1);
       }
       catch (DirectoryNotFoundException e)
       {
              Console.WriteLine("The file : " + sFilename +
                     " could not be found (" + e.Message + ")");
              Environment.Exit(-1);
       }
       catch (IOException e)
       {
              Console.WriteLine("Error reading the file: " + sFilename +
                     " (" + e.Message + ")");
              Environment.Exit(-1);
       }
       catch (OutOfMemoryException e)
       {
              Console.WriteLine("Not enough memory to read: " + sFilename +
                     " (" + e.Message + ")");
              Environment.Exit(-1);
       }
#endregion
```

Figure 3: Parsing the Command line arguments

Now that this is out of the way we can use the Add-in to select a SBW simulator to simulate the model. Select Tools\SBW Integration (see Figure 1: successful installation of the SBW Integration). The following Dialog will pop up:

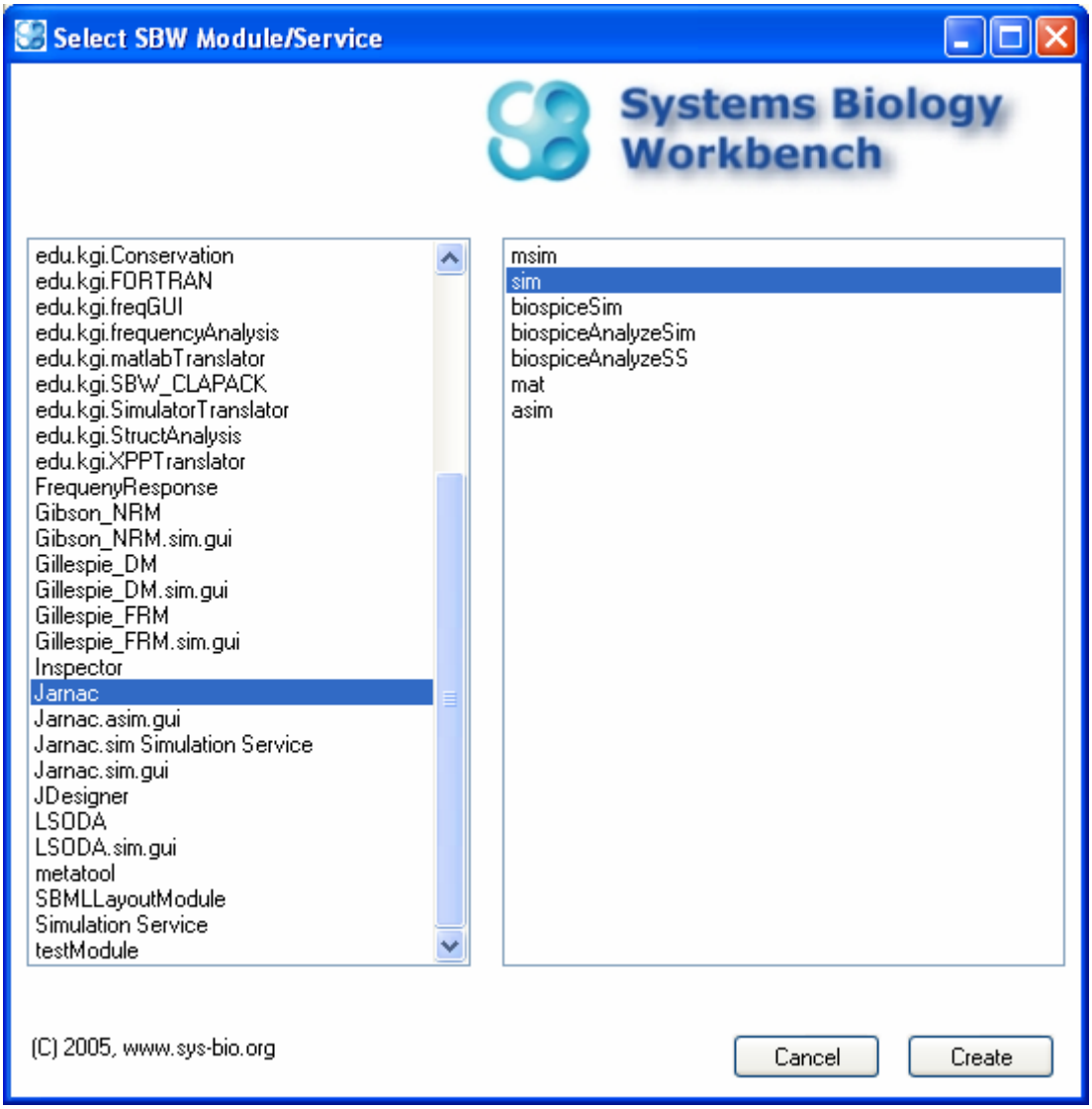


Figure 4: Selecting a SBW Module/ Service to wrap

Just select the SBW Simulator and the Service to wrap. Under Win32 this could be the Module “Jarnac” and the Service “sim”. If the assembly might be used on Linux/Mac, select a Module that is available on these platforms (for example “CVODE” is distributed with SBW on these platforms).

Upon a click on “Create” the next Dialog appears:

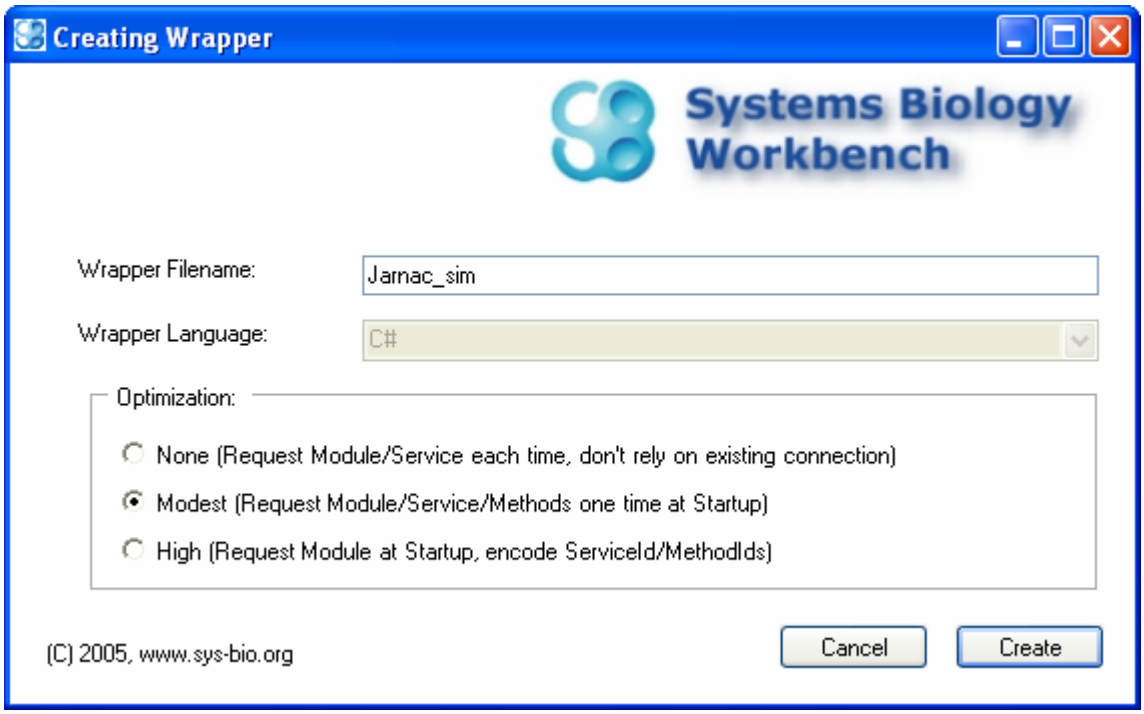


Figure 5: Options for the Wrapper

The TextBox “Wrapper Filename” allows modifying the filename of the wrapper that will be created in your project. “Wrapper Language” displays the Language of the Project that was detected by the Add-in.

The “optimization” section allows specifying how many SBW calls will be made from the Wrapper:

- **None:** In this mode the wrapper will request the Module/Service and Method each time a call is made. This results in a lot of SBW calls. Furthermore this method will reestablish a connection to the module in case of a disconnection, or unexpected shutdown of the module.
- **Modest:** Here the Module, Service and Method information will be requested from SBW only once upon the creation of the wrapper object. This method will be a lot faster during the method calls.
- **High:** Only the Module will be requested upon initialization of the Wrapper. Service and Method information will be provided by the Add-in. While this allows for a faster creation of the Wrapper object, the Wrapper might stop working with future versions of the module.

Upon a click on “Create” the wrapper code will be created and a new File will be added to your solution. Once that is done we can simply perform a simulation run like this:

```
try
      {
      Jarnac.sim.loadSBML     (sSBML);    // load the sbml
      Jarnac.sim.setTimeStart (0.0);      // simulate from t0 = 0
      Jarnac.sim.setTimeEnd   (1.0);      //          to   t1 = 1
      Jarnac.sim.setNumPoints (1000);     //          with 1000 steps

      // simulate the model
      double[][] oResult = Jarnac.sim.simulate();

      // print the result with 15 digit precision
      for ( int x = 0; x < oResult.Length; x++)
      {
            for ( int y = 0; y < oResult[x].Length; y++)
                  Console.Write( oResult[x][y].ToString( "F15")
                        + "\t");
            Console.WriteLine();
      }
}
catch (SBW.SBWApplicationException e)
{
      Console.WriteLine("Error during simulation: " + e.ToString());
      Environment.Exit(-2);
}

// disconnect from SBW ...
SBW.SBWLowLevel.disconnect();
```

Figure 6: C# simulating a SBML model

The wrapper implements every call to the SBW module as public static (public shared for VB) method. Hence no Object has to be created to call a module. (This reflects the stateless SBW API, if a Module requires a state it will either implement this behavior via different parameters or via a protocol that defines the order in which methods can be called.)

The last call in “Figure 6: C# simulating a SBML model” represents the disconnection from SBW. This should be done at the end of the program. If it is omitted the application will not terminate as it listens for possible incoming calls.

Known Limitations / Troubleshooting

- The Add-in currently does not allow for optimizations in a C++ Project.

- A VS 2005 Add-in will be released as soon as VS2005 is publicly available.

- Sometimes after an installation the “SBW Integration” entry does not appear. To solve this problem try:
  - Starting the “Visual Studio .NET 2003 Command Prompt”
  - And typing: devenv /setup

  or reinstall the Add-in.

- Sometimes after an uninstallation the “SBW Integration” entry does not disappear. To solve this problem try:
  - Starting the “Visual Studio .NET 2003 Command Prompt”
  - And typing: devenv /setup

- The program using the SBW C# bindings (or the Add-in) does not close at the end of Main. To solve this problem:
  - Add “SBW.SBWLowLevel.disconnect();” to the end of your code.