Troubleshooting after building

Building an application with SmartAssembly can cause three main types of problem:

- The application doesn't work
- SmartAssembly has not obfuscated enough of your code.
- Some dependencies are not merged or embedded.

This page explains why these problems can occur, and how to solve them.

Your application doesn't work after building with SmartAssembly

If you've enabled Method parent obfuscation, disable method parent obfuscation and try again. Method parent obfuscation significantly alters your assembly, and it doesn't work with all assemblies.

If your assembly has a *.exe.config* file, make sure that the *.exe.config* file is copied to the output directory.

To identify the cause of the problem, if the problem is not in a DLL:

1. Enable error reporting.
2. Rebuild the assembly.
3. Run the application again.
4. Use the error report to identify the line of your code where the problem occurs.

To identify the cause of the problem, if the problem is in a DLL:

1. Make sure that obfuscation is set so that you can decode any obfuscated symbols in error messages.
   To do this, go to the **Project settings**, and go to **Obfuscation**. Select **I want to obfuscate using only ASCII characters** and **I want to use a one-to-one renaming scheme**.
2. Rebuild the assembly.
3. Run the application again.
4. Note the ASCII obfuscated name in the error message.
5. Type the obfuscated name into the Decode Stack Trace feature in SmartAssembly.

When you've identified the line where the error occurs, check whether the following technologies are used at that line. If they are, see below for the appropriate solution:

- Using reflection to call a method or type by name
- Serialization
- Referencing code that has been pruned
- Enums used in a merged WPF dependency

In other cases, contact support for further guidance.

Using reflection to call a method or type by name

Obfuscation by name mangling renames methods and types. Normally, any given method or type will be renamed the same way throughout the SmartAssembly project, otherwise no application would work.

If you use reflection to call a method or type by name, the reference might not be renamed, and so the application will not work.

For example, imagine a simple calculator application, in which the Add() method is in a DLL called from the executable. After obfuscation, the Add() method is renamed to #KXf but the executable still contains the following line:

```
long sum = (long)addType.InvokeMember("Add", BindingFlags.InvokeMethod |  BindingFlags.Instance | BindingFlags.
NonPublic, null, addInstance, new object[] { num1, num2 });
```

After you've used error reporting to identify where the problem occurs, you can solve the problem:

- If the method you are calling is in a dependency, exclude that method from obfuscation. To do this:
  1. In the **Project settings**, go to **Obfuscation**.
  2. Next to the dependency name, click **Exclusions...**
  3. Navigate to the method.
  4. Select **Exclude the whole class (its definition and all its members)**.
  5. Click **Exclude from obfuscation**.
- You can also exclude the method from obfuscation using the `[DoNotObfuscate]` attribute. For more information, see Using custom attributes.
- If you want do not want to exclude the method from obfuscation, refactor your code to avoid calling the method or type by name by reflection.

Reflection works with obfuscation if you do not use names.

Serialization

The problem described above can also affect serialization. When a serialized object is unserialized, reflection is used to match the member's name in the serialized object to a name specified in the code.

- If the object is unserialized in the same assembly as the one where it was serialized, this is usually not a problem.
  If an error occurs, ensure that you have used the `[Serializable]` Attribute on all serializable classes, including classes that are XML serialized. For more information, see Using custom attributes.
- If the serialized object is unserialized into a different assembly, it will not work.
  Avoid writing obfuscated, serialized objects to files that might be opened in different builds of your assembly.

For more information about why serialization exceptions may occur, see Serialization exceptions occurring in obfuscated assemblies.

Referencing code that has been pruned

When you use SmartAssembly to prune code, it normally only removes code that is never called from other parts of the code. However, sometimes code is useful even if it is not called from anywhere else.

An example of this is WPF, where code might be referenced from the XAML file, but not from other parts of the code. SmartAssembly might prune this code, even though it is needed.

After you have used error reporting to identify where the problem occurs, exclude that code from pruning. For more details see Pruning unused code.

Enums used in a merged WPF dependency

Enums in merged dependencies containing WPF code might not be found. If this happens, you can either:

- use the `[DoNotObfuscate]`attribute to exclude the enum from obfuscation
  For more information, see Using custom attributes
- use the `<DoNotObfuscateAnyEnums/>` option in your **.saproj*file to exclude all enums in the assembly from obfuscation
  For more information, see the 'Using strings encoding with enums' section in Encoding strings

SmartAssembly has not obfuscated enough of your code

After building, if you inspect your assembly using .NET Reflector or similar, you may find that some code that you expected to be obfuscated has not been obfuscated by SmartAssembly.

This is because SmartAssembly will not obfuscate code if it determines that obfuscation might cause your application not to work, or that a string might be displayed in the user interface.

To see why SmartAssembly has not obfuscated your code, you can create a log file in 'TRACE' mode. For more information, see Log file for SmartAssembly.

The following types are always excluded from obfuscation to ensure that your application works correctly:

- Public members of DLLs
- Types with the `Serializable` attribute.
- Types with a base class of `System.MulticastDelegate`.
- Types with `System.ServiceModel.OperationContractAttribute` specified.
- Types with `System.ServiceModel.ServiceContractAttribute` specified.
- Types with any attribute starting `System.Xml.Serialization`.
- Methods with `System.Reflection.DefaultMemberAttribute` specified.
- Some types involved in XAML-binding

In addition, SmartAssembly might not obfuscate some of your types for the reasons described below:

- **Enums** are normally obfuscated, but SmartAssembly assumes that enum values which have `Format`, `GetName`, `GetNames`, `Parse`, or `ToStrin`
  called on them will be displayed to users. These enums are therefore excluded from obfuscation automatically.
- If a **member name** is the same in one more than one class, the member names might not be renamed.
  To ensure that they are renamed, under **Obfuscation**, select the **Advanced renaming** fields name mangling option.
  Note that selecting Advanced renaming prevents Decode Stack Trace from retrieving the names of the original fields.

Some dependencies are not merged or embedded

To avoid stopping your assembly from working, SmartAssembly does not merge or embed the following dependencies:

- DLLs containing a mixture of managed and unmanaged code.
- Primary Interop Assemblies.
- Any assembly signed using Microsoft's strong name key.

If you want to merge or embed a Microsoft assembly that is not a Framework Class Library, a workaround is described in Troubleshooting merging problems.