



Review SmartAssembly

Motivation

The tool SmartAssembly has often been mentioned to me as a good suitability for searching disguise- and error-reporting-tools. But up to now it hasn't been possible to have a far-reaching look at it.

My interest has been waken up again, when Jeff from GWB offered me to test SmartAssembly within the scope of the GWB-influencer program. After all this the motive was to get under way.

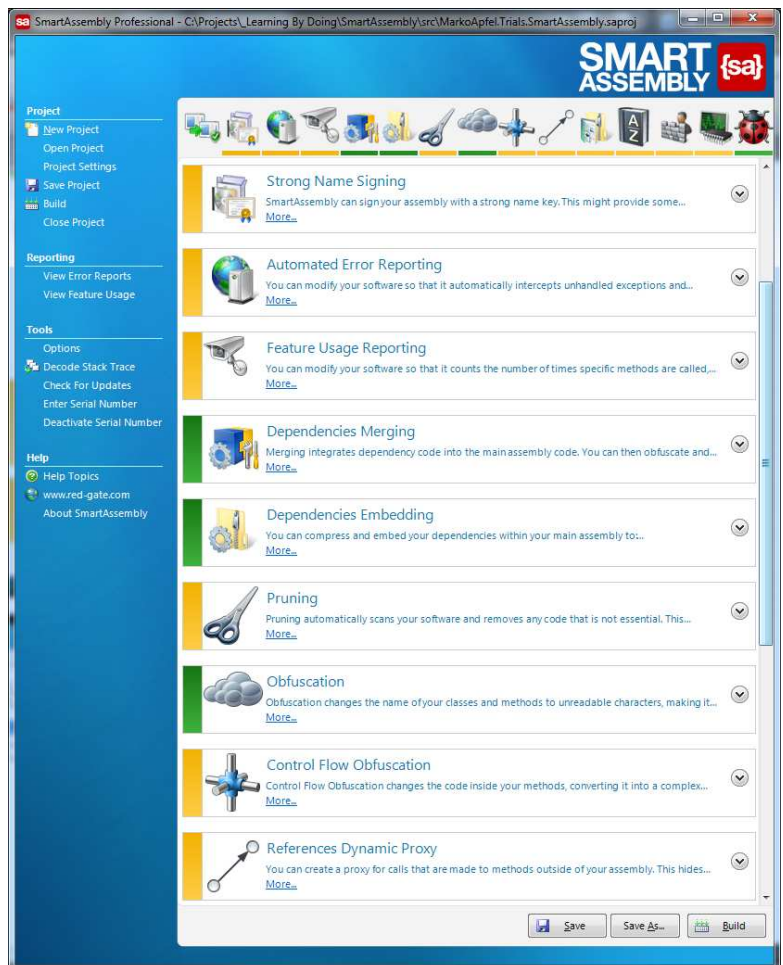
Installation

As a result Ben from Red Gate sent me the download- and licence data and the installation could have begun. For that purpose I have downloaded SmartAssembly Pro in the version 6.2 from <http://downloads.red-gate.com/SmartAssembly.exe> and started it under the credentials of the local administrator.

No further words are necessary for the setup – it directly works as you expect.

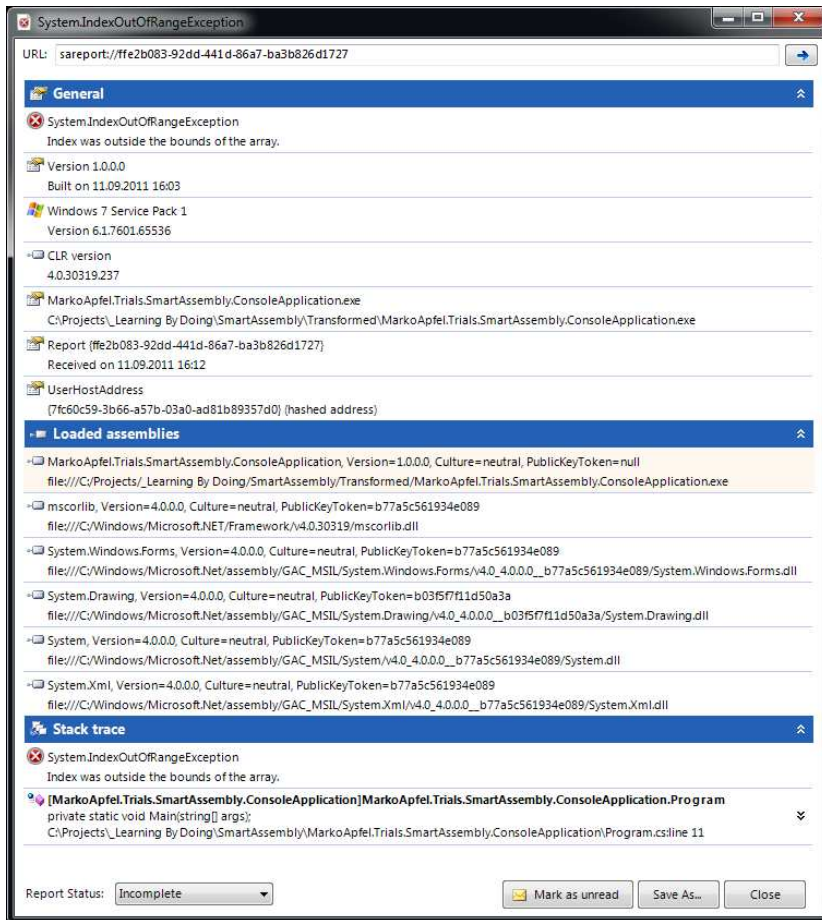
First steps

After starting SmartAssembly you will be met by one of the newfangled problem-orientated surfaces and you will be compelled to start a first project, as it were. Therefore I have created and compiled a completely simple Hello-World-Console-Application. Then this compilation was chosen in SmartAssembly and declared where the transformed assembly should be stored. Then a new list opens with tasks of such a variety I've absolutely never expected (see Screenshot).



Strong Name Signing

This choice not only allows the selection of the using keyfile, but also the installation of a new one, and so it forms a graphic alternative to the sn-tool.



Automated Error-Reporting

In the meantime everybody knows the error-reporting functions of Windows and other programs. SmartAssembly allows the embedding of such logic. For that purpose any method will be wrapped into an own try-catch-block and in the catch part the sending of information will happen.

This information seems to be stored on a server of Red Gate and can be studied by SmartAssembly in the menu point "View Error Reports".

For that purpose an abundance of information is being collected: Version, build date and place of running the assembly, Windows-version with patchlevel, CLR-information, further loaded assemblies and the stacktrace.

The reporting is clearly arranged by possibility of grouping, filtering and flagging – and it corresponds to the level of usual messaging-tools as mailers or feed readers.

Feature Usage Reporting

This function can track the use of features. This tracking is made on method-level by marking with the attribute ReportUsage. At the beginning of all such marked methods the tracker is called up and appropriate information are sent to a server of Red Gate, such as in "Automated Error Reporting".

You can see the information with SmartAssembly in the menu point "View Feature Usage". Apart from the call up of the method additional information for the carrying out platform are shown there – for example: Culture, Bitness, Windows-Edition.

Dependencies Merging

For getting the overhead under control, which is caused by loading of an assembly, you can mix together several assemblies. The standard tool for this is IL-Merge – a console tool. However, by using IL-Merge you have to realize which dependencies exist, for instance, to mix a whole set of matching assemblies. SmartAssembly gives the possibility of an automatic analysis and you can freely decide which assemblies should flow into the new one.

Dependencies Embedding

As by merging the types of a referenced assembly are mixed together in a new assembly, in the case of dependencies embedding the referenced assemblies are being coded as a whole and are being embedded as resource. During the running time a special static assembly-resolver-mechanism ensured the embedded assemblies to be deciphered and its return to CLR.

Also here is an analysis with Red Gates' own Disassembler Reflector shows an embedding of complex additional logic for managing.

Pruning

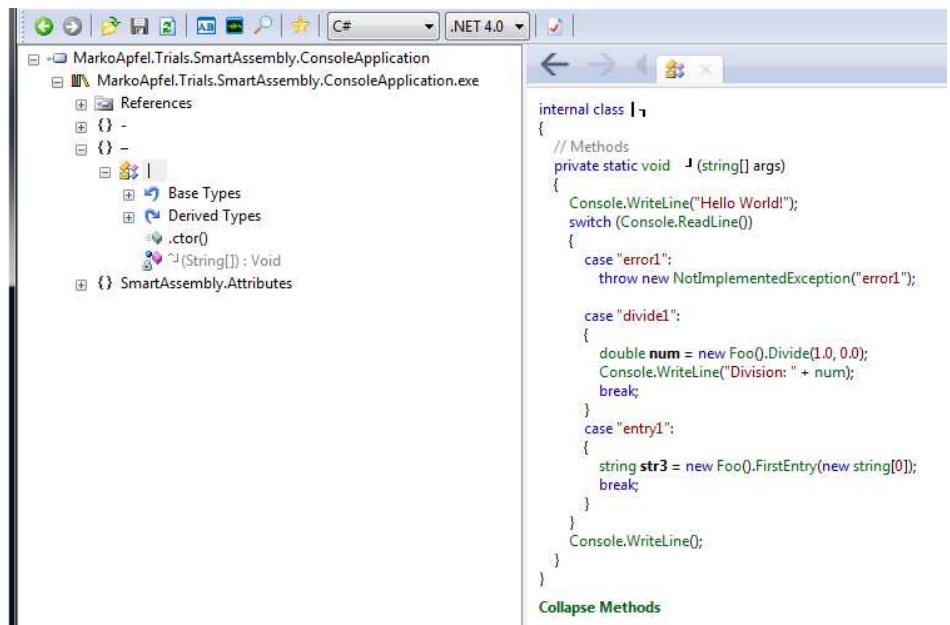
By pruning not essential components will be removed from the assembly. Those are, for instance, unnecessary metadata and unused code components. By using extended settings the pruning can be parameterized so that special types and name spaces are not being considered in pruning and consequently being maintained.

Obfuscation

This is probably the most known of SmartAssembly's features – disguising of code. By using the product, others will find it very difficult to have a look at the code after a disassembling and to understand it. For that purpose, among other things, the names of various users are being replaced by names with special characters, which representation is not possible for many editors and which for the user doesn't refer to a realized logic.

Especially the standard setting "I want to obfuscate using only Unicode unprintable characters" fulfills its name.

After obfuscating my demo code the names of namespaces as well as of methods are completely incomprehensible and cryptic.



Control Flow Obfuscation

Such a disguised code should mix up the IL-Code and make it to a spaghetti code so that decompilers resign. And really the Red Gate's own reflector puts out only one remark.:
//This item is obfuscated and can not be translated.

So far, so good. But what do the decompiler competitors say? The (up to now) free alternative dotPeek from JetBrains is able to decompile the code and, in fact, as it was in the original – at least in the standard setting "Fastest". In the setting "Strongest" at least a code is being created, which can hardly be followed by many label- and other jumps. However, it has been decompiled.

References Dynamic Proxy

This option weaves code in the assembly which produces dynamic proxies for calls outward during its running time. By cracking the assembly these proxies are being impeded in their function, so that an attempted bending of calls leads to mistakes.

Resources Compression and Encryption

With that resources are being coded and compressed. During the running time the additional logic makes sure that decoding and decompressing will happen only one time at the first call up and will be cached from now on.

Strings Encoding

Instead of coding the resource information as a whole, only the strings will be coded by this option.

By coding the string-resources by one of both ways a good beginning for re-engineering of the program logic is being destroyed. You haven't got a relation to UI-feedback and code. Moreover, sensitive information such as SQL-queries, licence numbers, generic passwords and others are being protected by these codes.

Other Protections

By invalid metadata or using the SuppressIldasm-Attribute respectively a decompiling should be made more difficult.

However, both variants of protection don't disturb the reflector.

Other Optimizations

Through memory optimizing and extensive using of the sealed-marking the application can be performed.

Generate Debugging Information

This creates a PDB-File for debugging a transformed assembly.

Of course, most of the named functions can be mixed, for instance, in order to get a single disguised assembly for delivery by Dependencies Merging with following obfuscation.

Further options

Mark as released

Without these markings information to error- and feature-reports will only be stored for 180 days. By that the reports accumulated during the development can be tidied up.

Summary by Build

After transforming the realized steps are being shown clearly.

Storage for path specifications in the project

In the sense of CI-using all artefacts below a DevTree should be accessible via relative paths. As far as that goes the choice of the absolute path specifications must be handled very carefully.

Reports Database Options

For collecting information for reportings by a client a local access- or a SQL Server Databank from network can be brought in action.

CI-server integration

Red Gate supplies a console-based program in order to carry out all these steps automatically. This forms the basis for CI-processes. With JetBrains TeamCity only a Build Step with a call up of this program is necessary.

Build Steps

There are **2** build steps defined.

Build Step	Description
Build Project SmartAssembly	MSBuild Build file: .\SmartAssembly\src\MarkoApfel.Trials.SmartAssembly.sln Targets: default
Obfuscate	Command Line Command: .\SmartAssembly\tools\SmartAssembly\SmartAssembly.com .\SmartAssembly\src\MarkoApfel.Trials.SmartAssembly.sproj

Conclusion

SmartAssembly could be the perfect base to protect your intellectual property and optimize your assembly for management and performance questions.

Marko Apfel

<http://geekswithblogs.net/mapfel/>