NAME

Mono CIL Linker

SYNOPSIS

monolinker [-o output_directory][-l i18n_assemblies][-c skip | copy | link] -x descriptor | -a assembly | -i info_file ...

DESCRIPTION

monolinker is a CIL Linker. The linker is a tool one can use to only ship the minimal possible set of functions that a set of programs might require to run as opposed to the full libraries.

The linker analyses the intermediate code (CIL) produced by every compiler targeting the Mono platform like mcs, gmcs, vbnc, booc or others. It will walk through all the code that it is given to it, and remove all the unused methods and classes. This is done using a mark and sweep operation on all the code that it is referenced.

The generated output from the monolinker can be later processed by the *mkbundle* tool to generate small native self-contained executables.

Do not confuse this with the Assembly Linker (al) which creates assemblies from manifests, modules and resource files.

OPTIONS

-d search_directory
: Specify a directory to the linker where to look for assemblies.

-o output_directory
: Specify the output directory, default is 'output'.

: If you specify the directory '.', please ensure that you won't write over important assemblies of yours.

-b true | false
: Specify whether to generate debug symbols or not, default is false.

-g true | false
: Specify whether to generate a new guid for each linked module or reuse the existing one, default is true.

-l i18n_assemblies
: Specify what to do with the region specific assemblies

: Mono have a few assemblies which contains everything region specific:

```
I18N.CJK.dll
I18N.MidEast.dll
I18N.Other.dll
I18N.Rare.dll
I18N.West.dll
```

: By default, they will all be copied to the output directory, but you can specify which one you want using this command. The choice can either be: none, all, cjk, mideast, other, rare or west. You can combine the values with a comma.

-c action
: Specify the action to apply to the core assemblies.

: Core assemblies are the assemblies that belongs to the base class library, like mscorlib.dll, System.dll or System.Windows.Forms.dll.

: The linker supports three operations on these assemblies, you can specify one of the following actions:

skip	This instructs the linker to skip them and do nothing with them.
copy	This instructs the linker to copy them to the output directory,
link	This instructs the linker to apply the linking process and reduce their size.

-p action assembly
Specify per assembly which action to apply.

-x descriptor
Use an XML descriptor as a source for the linker.

Here is an example that shows all the possibilities of this format:

```
<linker>
        <assembly fullname="Library">
                <type fullname="Foo" />
                <type fullname="Bar" preserve="nothing" required="false" />
                <type fullname="Baz" preserve="fields" required="false" />
                <type fullname="Gazonk">
                        <method signature="System.Void .ctor(System.String)" />
                        <field signature="System.String _blah" />
                        <field name="someFieldName" />
                </type>
        </assembly>
</linker>
```

In this example, the linker will link the types Foo, Bar, Baz and Gazonk.

The preserve attribute ensures that all the fields of the type Baz will be always be linked, not matter if they are used or not, but that neither the fields or the methods of Bar will be linked if they are not used. Not specifying a preserve attribute implies that we are preserving everything in the specified type.

The required attribute specifies that if the type is not marked, during the mark operation, it will not be linked.

The type Gazonk will be linked, as well as its constructor taking a string as a parameter, and it's _blah field.

You can have multiple assembly nodes.

-a assemblies
use an assembly as a source for the linker.

The linker will walk through all the methods of the assembly to generate only what is necessary for this assembly to run.

-i info_file
use a .info xml file as a source for the linker.

An info file is a file produced by the tool mono-api-info. The linker will use it to generate an assembly that contains only what the public API defined in the info file needs.

-s [StepBefore:]StepFullName,StepAssembly[:StepAfter]

You can ask the linker to execute custom steps by using the -s command. This command takes the standard TypeFullName,Assembly format to locate the step. You can customize its position in the pipeline by either adding it before a step, or after.

Example:

```
using System;
using Mono.Linker;
using Mono.Linker.Steps;

namespace Foo {

        public class FooStep : IStep {

                public void Process (LinkContext context)
                {
                        foreach (IStep step in context.Pipeline.GetSteps ()) {
                                Console.WriteLine (step.GetType ().Name);
                        }
                }
        }
}
```

If you compile this custom against monolinker to a Foo.dll assembly, you can use the *-s* switch as follows. To add the FooStep at the end of the pipeline:

```
monolinker -s Foo.FooStep,Foo -a program.exe
```

This commanand will add the FooStep after the MarkStep:

```
monolinker -s MarkStep:Foo.FooStep,Foo -a program.exe
```

This command will add the FooStep before the MarkStep:

```
monolinker -s Foo.FooStep,Foo:MarkStep -a program.exe
```

This command will add the FooStep before the MarkStep

-m CustomParam ParamValue
Specify a parameter for a custom step.

COPYRIGHT

BUGS

Bugs report are welcome at https://github.com/mono/linker/issues

Product Mono Tools, Component linker.

MAILING LISTS

Mailing lists are listed at http://www.mono-project.com/community/help/mailing-lists/

WEB SITE

http://www.mono-project.com/docs/tools+libraries/tools/linker/

AUTHORS

The linker has been written by Jb Evain, and have been partially founded by the Google Summer of Code.

LICENSE

The linker is licensed under the MIT/X11 license. Please read the accompayning MIT.X11 file for details.

SEE ALSO

al(1),mkbundle(1),mono(1),mcs(1).