CDN2 : CCRefGen Design

# Overview

CCRefGen’s purpose is to produce a *contract reference assembly* for a given assembly A. By convention, the contract reference assembly will be named A.Contracts.dll, irrespective of the extension of A. We will use A as the input assembly and A.Contracts.dll as the output of CCRefGen in the remainder of the document.

# Transformation

To a first approximation, CCRefGen produces an assembly that is almost identical to the input assembly, but where the code inside method bodies has been reduced to the code specifying CodeContracts.

CodeContracts are recognized by the format described in CDN0: Contract Format.

## Three Argument Contract Methods

CCRefGen normalizes all uses of CodeContract methods to a format where a CodeContract method takes 3 arguments:

1. The Boolean condition
2. A user defined string
3. The original source text of the condition

It does so by extracting the source text from the original source files using PDB information and changing any calls to CodeContract methods (such as Requires, Ensures, etc.) to the three-argument version. If no user-provided string appears in contract calls, the null literal is used as the second argument, and if no source string can be extracted, a null literal is used for the third argument.

Since the CodeContract library does not contain such three-argument versions of the contract methods, CCRefGen emits the following new type and methods into the output dll;

class \_\_ThreeArgumentVersions {

}

# Tree Shaking

In principle, CCRefGen can remove all metadata and associated code that is not referenced from any contract code. Care must be taken to make sure that types, methods, and code associated with anonymous delegates used in Contract.Forall or Contract.Exists are properly maintained in the contract reference assembly, as such code must be inheritable across assemblies.

The following pseudo algorithm determines for each metadata entity whether it needs to be included in the output. Additionally, for methods, it determines if the original code body needs to be included.

Output: Set<Member> toInclude;

Set<Method> bodyToInclude;

WorkList<Member> pending;

Initialization:

Foreach type t in A visible outside A

Add t to toInclude

Add t to pending

While (pending is not empty)

p = pull from pending

if (p is a type)

forach member m in p visible outside A

if (m is not in toInclude) {

add m to toInclude

add m to pending

}

endif

## Examples

### Requires

The following requires calls

Contract.Requires(x != null);

Contract.Requires(y > 0, “y must be positive”);

Contract.Requires<NullArgumentException>(z != null);

Contract.Requires<ArgumentException)( q >= 0, “non-negative”);

Are transformed by CCRefGen into calls

\_\_ThreeArgVersions.Requires(x != null, null, “x != null”);

\_\_ThreeArgVersions.Requires(y > 0, “y must be positive”, “y > 0”);

\_\_ThreeArgVersions.Requires<NullArgumentException>(z != null, null, “z != null”);

\_\_ThreeArgVersions.Requires<ArgumentException)( q >= 0, “non-negative”, “q >= 0”);

Similarly, calls to Ensures, and Invariant are converted to their three argument versions.