Contract Reference Assembly Service

The contract reference assembly service is a mechanism for updating the contracts associated with libraries, for reviewing such updates, and for distributing updated contract reference assemblies (CRAs).

# Vision

Upon discovering that a method M has a missing contract, a user clicks on “goto definition” or F12 to bring up the metadata interface of the method. An editor extension displays the current contracts on the method and also allows entering/editing the contracts in source form directly inside the window of Visual Studio (can we use Roslyn for this?).

Submitting/saving such changes will send them to a centralized server, initially hosted by MSR, where they are added to the contract repository. An automated build validates the changes and if acceptable, new reference assemblies are built automatically and made available for download. Updated contract reference assemblies can be downloaded either manually and installed over the existing ones, or the contract tools can automatically download updated version with permission from the user.

Users submitting such changes need to use a Windows Live ID for identification. A website (codecontracts.microsoft.com) hosted by the server allows a user to see whether submitted changes were accepted or rejected and the reasons thereof. An administrator (initially MSR folk) is the final arbitrator on what contracts are accepted. Methods can be locked down to prevent further updates when the contracts are deemed complete.

# Prerequisites

The current installer contains both the contract tools and the contract reference assemblies. We need to split off the contract reference assemblies so that they can be updated separately and have a separate version number. We need to have fixed version numbers for snapshots of contract reference assemblies in order to reproduce customer issues.

# Contract Change Units

Each method contract change submitted represents a contract change unit (CCU). A CCU initially enters a submitted queue. The status of a CCU is akin to the queue it is in.

The contract reference service contains a number of distinct queues:

* **Submitted** (contains CCUs not yet tested/reviewed)
* **Rejected** (CCUs that will not be further reviewed, comments indicate why the CCU was rejected)
* **ToReview** (CCUs that need to be reviewed by administrator)
* **Pending** (CCUs that passed tests and are waiting to be included in the next build)
* **Built** (CCUs that pass tests and are currently included in the built reference assemblies)
* **Approved** (CCUs that were permanently approved. They might be superseded by later changes on the same contract.)

The service monitors the **Submitted** queue. If a new item is available, the system determines if the change is on the currently last approved contract for the method. If it isn’t, the change needs to be resubmitted as a delta on the most recent approved change and enters the rejected queue. Otherwise the change is applied to the appropriate method contract and a contract reference assembly is built including the change. The contract is validated against the real reference assemblies to see which target framework versions the change is valid for. The change is also validated against objectionable content. If the build fails or any validations fail, the CCU moves to the **Rejected** queue and is tagged with the reasons for the rejection.

Then the change is reviewed for other acceptance criteria, including, but not limited to determining if the change is on a locked down method. If the CCU fails auto-acceptance, the CCU moves to the **ToReview** queue.

Otherwise, the CCU enters the **Pending** queue.

The service also monitors the **pending** queue and starts building new contract reference assemblies upon changes in this queue. Each pending CCU is reviewed to see if it conflicts with a CCU in the built queue. Two CCUs conflict if they affect the same method, and the later change is not derived from a build that includes the earlier change. This means that any particular build will contain at most a single new CCU per method. Non-conflicting changes are added to the Built queue, whereas conflicting changes are either put into the rejected queue (if the published contracts contain the conflicting change), or kept in the pending queue. This way, it will get rejected as out-of-date when the new build is available.

New contract reference assemblies are built from built and approved changes.

CCUs may also move between queues based on administrator intervention. E.g., Built CCUs are reviewed for final inclusion into the Approved queue or moved to the rejected queue with an indication why the change was rejected. Administrators can also move CCUs in the ToReview queue to the Pending queue upon inspection.

Finally, users can inspect their rejected CCUs and decide to submit them for review.

# Technical Parts

Our current source based building of contract reference assemblies is cumbersome and does not scale well. We need a way to build contract reference assemblies from normal assemblies/reference assemblies and source snippets for contracts.

The alternative would be to use CCI2 or Roslyn to setup the method context for a method contract and to compile just the contract snippet in that context. The resulting IL would then be injected into the empty contract reference assembly generated from the original assembly.

This way, we should be able to get rid of the #ifdef use for compiling to different framework assemblies. Instead, we would simply try to compile the contract code for each framework context and inject it into the appropriate contract reference assembly.

The downside of this is that we don’t have a stand-alone source that can be built with standard tools. The upside is that we have one snippet per contract and automatic handling of framework versions.