Apollo

License verification specification

Confidential

Revision number: 1

Date:

Name: P. van der Velde

# Disclaimer

This specification is not, by any stretch of the imagination, complete. It will need to be revised several times before it is complete. Currently several major parts are either missing or incomplete. This disclaimer will be updated to reflect any change in these sections. Finally a specification is supposed to be a ‘living’ document and therefore never complete. What were you thinking, better learn to live with this fact.

# Introduction

# Implementation

* Create a verification class which receives the results of a verification. Each service can hold on to one of these. Verifications time out after XX time. When no valid verification is found then the service should stop functioning and send a request for new verification. If no verification is available within YY time then we stop and notify the user
* The verification class can also handle the requests for the encrypted code
* Which kind of key validation will we use? Partial verification sounds like a good plan
* How will we deal with an external license service?

Another thing to think about is the security messages. They should never be displayed directly. This is where the messaging system comes into play. First we have to send all messages for the user to the UI via the message service. We could include a key in the messages which indicates where to get the actual translated message (from a resource file). We’ll need to see how to do this though because we don’t want the UI to load all the assemblies. Only system assemblies should be loaded. So in that case we’ll have to send a message back into the AppDomain where the message came from and load the data there.

## Time based verification

* Run one or more thread based timers. Each timer has a random time interval with a small, medium or large time base (e.g. minutes, hours, days, weeks, months). When the timer fires we validate the license. Time can be based on last start-up, install date or other.
  + A hourly timer for the current run
  + A daily timer based on the last start-up set (maybe the last one in a set of 10, or a random set with a maximum size of 50). Maybe also include the last check date
  + A weekly timer based on the last start-up set (ditto to the daily one). Maybe even include the install date or the last check date
  + A monthly timer which is based on the install date + last check date
  + A random time timer which is based on the install date + last check date.
* The threads must be made bullet-proof so that injecting thread errors does not lead to a failure of the license system. A dying thread should lead to creating a new one and starting that.

## Encrypted code store

* Encrypt code during compilation. Only provide store with a public key which is based on the license key

The obvious questions are:

* How do we hide code in an encrypted block
* Which pieces of code will we hide
* How to determine that code can be decrypted / when etc.
* How do we make this safe so that the user can use it but not crack it ... mmmm this seems to be a paradox.