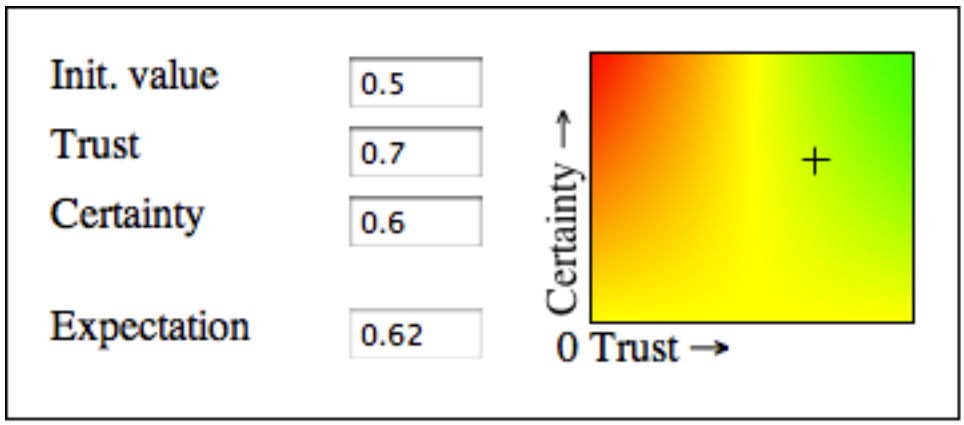
# CERTAIN TRUST API

*Tutorial*



# CERTAIN TRUST API

## Tutorial

## Introduction

What is Certain Trust API?

We often try to find information on the trustworthiness of a service we use or a product we buy, and the most common way to do it is to gather people's evidence of the previous experience, which afterwards we combine to build our own opinion. That is why it seems just a nice idea to provide a service, that would manage opinions, gathered from a great amount of sources, modify them, combine and display in a user-friendly way.

Certain Trust API provides these tools exactly. It is based on a Certain Trust model and has two classes: CertainLogic and CertainLogicHTI. The object of the first one represents trustworthiness of a subject or a system. It can be based on the single experience or derived from many gathered evidences. The second class displays the CertainLogic object in an intuitively understandable way (HTI stays for Human Trust Interface).

The API is currently implemented in two versions: JavaScript and Java.

## Getting started

Start by creating a CertainLogic object. This is the most common constructor:

```
CertainLogic(double t, double c, double f, int n);
```

in which you can pre-install the values of the rating $t$ and the certainty of this rating $c$. You should also set the values $f$ (the initial trust) and $N$ (the maximal number of expected evidence). $T$, $c$ and f vary from 0 to 1, with 0 being very negative and 1 - very positive. $N$ is a positive integer value, which typically goes from 5-10 to 100 or even 1000, but in fact has no upper limit.

As an example:

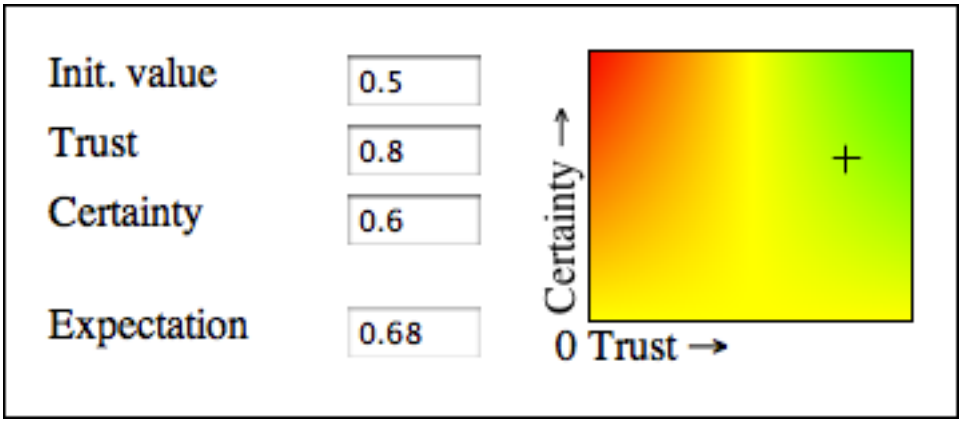
```
BookstoreRating = CertainLogic(0.8, 0.6, 0.5, 10);
```

It means, that the average opinion about this bookstore is pretty positive ($t$ = 0.8), that there is no initial opinion about this shop ($f$ = 0.5 is neutral). $N$ = 10 means, that we need at least 10 customer reviews to make an adequate and trustworthy rating of any bookshop. The certainty is not very high (c = 0.6), so probably this whole rating was based on 5-6 opinions left by customers.

Now to display this rating you simply have to create a CertainLogicHTI object:

```
CertainLogicHTI(BookstoreRating);
```

and place it on your web page or in your application.




The visualization of BookstoreRating

## Certain Trust model and Bayesian representation

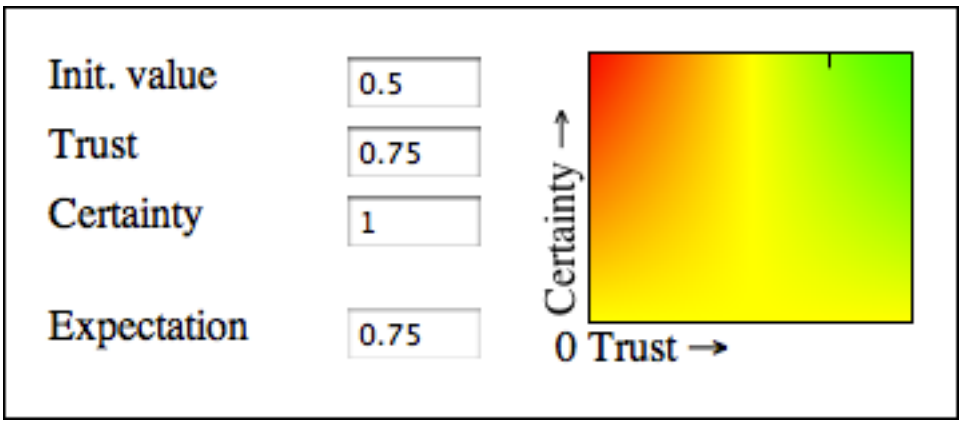
Certain Trust model is itself an extension of Bayesian approach and therefore has a connection with it. That's why there is another constructor, that uses the Bayesian representation:

```
CertainLogic(double r, double s, int n);
```

where $r$ is a number of positive evidence, while $s$ is negative. As an example:

```
OnlineLibrary = CertainLogic(15, 5, 20);
```

the values of $r/s$ representation will be mapped into the Certain Trust model and the object OnlineLibrary could be used as usually.




The visualization of OnlineLibrary

There is one more constructor:

```
CertainLogic(int n);
```

which you can use to create a neutral element.

## Modifying an Opinion

After an object of CertainLogic class have been created, you have an opportunity to change its parameters. Use one of these functions:

```
void setN(int n);
void setF(double f);
void setTC(double t, double c);
void setRS(double r, double s);
```

Any of these changes may cause automatic update of the dependable values, for example change of positive/negative evidences ($r$ and $s$) affects trust and certainty ($t$ and $s$), and the reset of $N$ will lead to corresponding adjustment of certainty, and probably $r$ and $s$ as well. All of this is done internally and you may be sure, that at any moment object's parameters do not contradict each other.

## Combining Opinions

Sometimes you want to evaluate the trustworthiness of a complex system. For example the quality of an online bookstore depends on how convenient and accessible their web page (A) is and how good their delivery system operates (B). Both of these components are important and necessary, so their trustworthiness would be combined with an AND operator. Access to the web page is however ensured by three servers (S1, S2 and S3) - one of which is enough to handle the task, so their evaluation should be put together with an OR operator. CertainLogic provides both of these functions:

```
1.                A = S1.OR(S2, S3);
2.  BookstoreRating = A.OR(B);
```

The number of arguments is unlimited. The only restriction is, that all the objects must have the same $N$ value.

If necessary, you can negate an opinion with a NOT operator:

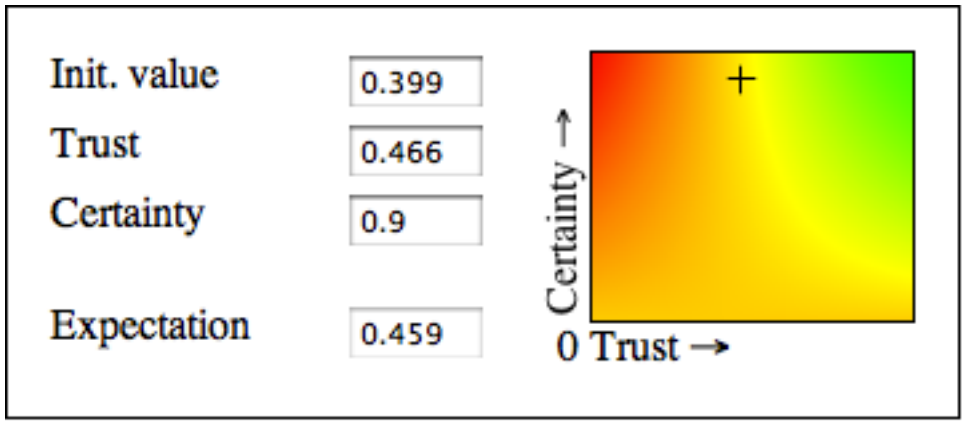
```
S = S.NOT();
```

Another way to combine opinions is a FUSION function. For example, you have gathered 10 ratings on one subject or item. Use fusion to derive an average of these ten. Fusion function has two versions: weighted and conflicted.

```
static CertainLogic wFusion(CertainLogic[] args, int[] weights);
static CertainLogic cFusion(CertainLogic[] args, int[] weights);
```

They both take two arrays of the same size, the first one holds opinions, and the second - their weights.

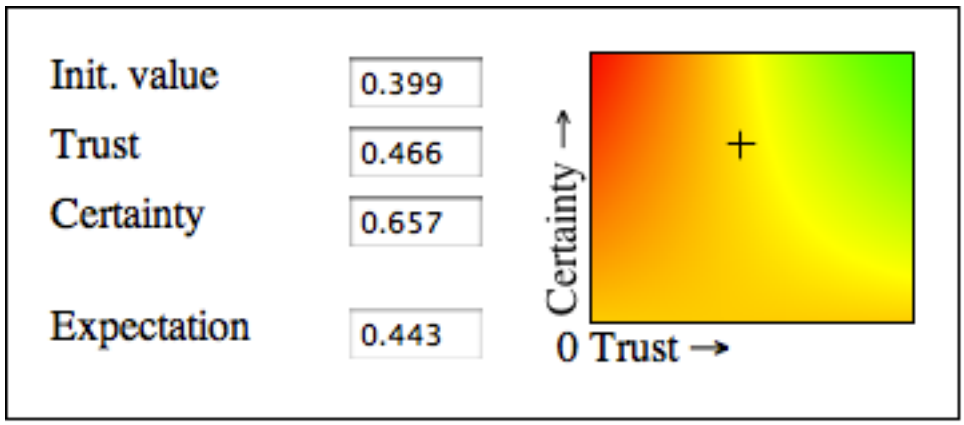
```
1. CertainLogic firstCustomer = new CertainLogic(0.8, 0.9, 0.5, 100);
2. CertainLogic secondCustomer = new CertainLogic(0.3, 0.9, 0.35, 100);
3. CertainLogic[] opinions = {a, b};
4. int[] weights = {2, 1};
5.
6. CertainLogic storeRating = CertainLogic.wFusion(opinions, weights);
```




wFusion

The only difference between wFusion and cFusion is that the latter also takes into consideration how conflicted the opinions are, and calculates the degree of conflict, which may then affect the certainty of a result.

```
1. CertainLogic firstCustomer = new CertainLogic(0.8, 0.9, 0.5, 100);
2. CertainLogic secondCustomer = new CertainLogic(0.3, 0.9, 0.35, 100);
3. CertainLogic[] opinions = {a, b};
4. int[] weights = {2, 1};
5.
6. CertainLogic storeRating = CertainLogic.cFusion(opinions, weights);
```




cFusion