

CertainTrust SDK: Programmer's Quick Start

An introduction into programming Java and JavaScript
with CertainTrust, the Human Trust Interface, and CertainLogic.
Use trust and reputation under uncertainty in your programs.



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Telecooperation Lab

This document is intended as a programmer's quick start tutorial to writing programs using the CertainTrust SDK with both Java and JavaScript.

Introductory notes to instantiating CertainTrust and CertainTrustHTI objects, their methods and the CertainLogic operators are accomplished by step-by-step tutorials which re-implement the supplied demonstrators.

More in-depth information can be found in the demonstrators that come with the CertainTrust SDK package and the sourcecode itself.

CertainTrust SDK: Programmer's Quick Start

Version 1.0

10 March 2013

Technische Universität Darmstadt

Telecooperation Lab

Hochschulstraße 10

64283 Darmstadt

Prof. Dr. Max Mühlhäuser

max@informatik.tu-darmstadt.de

Florian Volk

florian.volk@cased.de

David Kalnischkies

Maria Pelevina

<http://www.tk.informatik.tu-darmstadt.de>

Table of Contents

Table of Contents	1
1. Introduction to the Trust Model CertainTrust	2
1.1. <i>CertainTrust</i>	2
1.2. <i>Opinions</i>	3
1.3. <i>CertainLogic: A Logic for Reasoning under Uncertainty</i>	4
2. Core API Documentation	6
2.1. <i>The CertainTrust Class</i>	6
2.2. <i>CertainLogic Operators</i>	7
2.3. <i>The CertainTrustHTI Class</i>	8
3. Using Evidences in CertainTrust	11
4. Step-By-Step Guide: Java	12
4.1. <i>Importing the CertainTrust SDK JAR</i>	12
4.2. <i>Project Skeleton Code</i>	13
4.3. <i>Creating and Displaying a Human Trust Interface</i>	13
4.4. <i>Programmatically Accessing CertainTrust Data</i>	14
4.5. <i>Using CertainLogic</i>	15
5. Step-By-Step Guide: JavaScript	18
5.1. <i>Project Skeleton Code</i>	18
5.2. <i>Creating and Displaying a Human Trust Interface</i>	19
5.3. <i>Programmatically Accessing the CertainTrust Data</i>	19
5.4. <i>Using CertainLogic</i>	20
Acknowledgement	23

1. Introduction to the Trust Model *CertainTrust*

With IT systems becoming highly distributed and managed by multiple parties, the successful outcome of user interaction with them has become dependent not only on secure end-to-end connections to foreign infrastructures or services, but also on establishing their trustworthiness.

Whenever it is impossible to guarantee particular properties of a specific system behavior, e.g., due to lack of full knowledge or personal experience, soft security mechanisms such as computational trust come in play. Trust is an approximator for future behavior, used to make claims based on direct previous experience of a user, recommendations from third parties and possibly additional information such as social relationships or indicators of trustworthiness. Many trust models have been developed to derive this value and many of them successfully achieved established goals.

However, there are some issues still left underrepresented by the existing approaches, two of them being the following:

- Uncertainty associated with the trust value. Whereas the expectation about future behavior can be generally defined as a relation between positive and negative experiences, the concept of “lack of information” shouldn’t be neglected. Insufficient amount of evidence lowers the certainty of the trust value, which should itself influence the final expectation value
- Evaluation of complex systems, considering the trustworthiness of their subsystems and atomic components, according to their layout and internal dependencies.

1.1. *CertainTrust*

CertainTrust is a novel approach in the field of Computational Trust that addresses the aspects mentioned above and provides a user-friendly, ready to use, and consistent way for the evaluation of the trustworthiness of systems, eventually helping users in their decision making.

It is based on the Bayesian approach, taking big effort into supporting the uncertainty concept that ensures more realistic and precise estimation of the trust and ratings. Furthermore, it is completed with a range of standard operators of propositional logic, which are introduced in the next chapter. These operators cover most of the common dependencies and relations that can exist between atomic components or subsystems, thus providing effective tools to model trust evaluation for composite systems. This is especially important when the performance of a service highly depends on the reliability of its subcomponents.

Not less important for the adaptation of the trust model as a rating system is its visual representation. That is why a new representation called *Human Trust Interface* (HTI) was designed to display a CertainTrust opinion in an intuitively understandable way. It carries out the representational function, but also the interactive one: users can modify a CertainTrust opinion and immediately observe the changes.

The SDK covers two programming languages with the Java and JavaScript implementations. This way it has been made possible to integrate CertainTrust in both online and offline applications.

1.2. Opinions

The CertainTrust object models an opinion or a belief in the truth of a proposition. It is based on the idea, that for a good evaluation, not only a relation between available positive and negative evidences is necessary, but also the degree of certainty. The certainty is influenced by a variety of factors: it could fall low if the number of evidences is regarded to be too small to be representative (even when all of them are positive) or when they come from unreliable, suspicious third parties. If the certainty in the provided evidence is low, the evidence has less influence on the determination of the final belief.

The CertainTrust instance is represented by three parameters (t , c , f): *Trust*, *Certainty*, and *Initial Trust*. They have the following semantics:

- *Trust* $t \in [0, 1]$ indicates the degree to which past observations (if there are any) support the truth of the proposition. This value is calculated as a relation between the number of supporting evidences and the overall number of evidences gained.

Its extreme values can be interpreted in a following way:

Trust = 0: There is only evidence contradicting the proposition.

Trust = 1: There is only evidence supporting the proposition.

- *Certainty* $c \in [0, 1]$ indicates the degree to which the trust is considered to be representative for the future. The higher the certainty of an opinion is, the higher is the influence of the trust on the expectation value in relation to the *Initial Trust*. When the maximum level of certainty ($c = 1$) is reached, the *Trust* value is assumed to be representative for future outcomes.

Its extreme values can be interpreted in the following way:

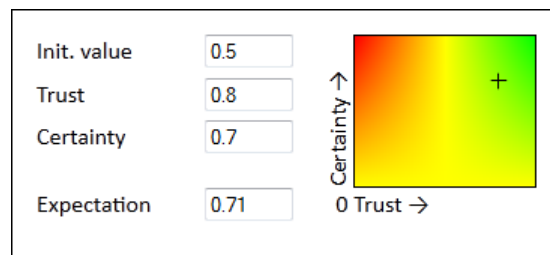
Certainty = 0: There is no evidence available.

Certainty = 1: The collected evidence is considered to be representative.

- The *Initial Trust* $f \in [0, 1]$ expresses the assumption about the truth of a proposition in absence of evidence.

As this Trust/Certainty view is in many ways based on the evidence-based view, both are compatible and interchangeable within the CertainTrust SDK. Apart from described (t, c, f) triple, the CertainTrust keeps, updates, and allows manipulating the corresponding values r and s (number of positive and negative evidences the opinion is built on). An additional parameter N denotes the amount of evidences considered to be sufficient to provide representative evaluation. As the provided amount of evidences approaches the value N , the certainty of an opinion rises.

The Human Trust Interface displays the *Expectation* about the truth of the proposition, calculated from *Trust*, *Certainty* and *Initial Trust*:



1.3. CertainLogic: A Logic for Reasoning under Uncertainty

It is common for the quality of a service to depend on the performance of its sub-elements. The layout of the system, as well as dependencies between its components, can be modeled in terms of propositional logic. And to evaluate the trustworthiness of such a composite system, based on the evidence about its atomic components, several operators have been developed.

The *CertainLogic* operators can be separated into two groups:

The first three (OR, AND, NOT) work with *independent opinions*. In this context it most usually means “on two different subjects” (e.g., one opinion on the latency of a service, another on its security). These operators can be used to build and evaluate systems that rely on many elements.

- The operator **OR** is applicable when opinions for two independent propositions form a new opinion reflecting the degree of truth for at least one out of both propositions. It is similar to Boolean logic’s OR: at least one positive opinion results in a positive opinion, otherwise a negative one.

-
- The operator **AND** is applicable when opinions for two independent propositions are aggregated to produce a new opinion reflecting the degree of truth of both propositions simultaneously. It is similar to Boolean logic's AND: at least one negative opinion leads to a negative opinion.
 - The operator **NOT** is applicable when an opinion about a proposition needs to be negated. It should be used whenever degrees of "No" are actually positive answers.

Another group consists of the operators **wFUSION** and **cFUSION**. These are intended to fuse multiple dependent opinions on the same subject into one (e.g., a group of n users tests the service under the same conditions and gives n individual opinions on its performance).

- Weighted fusion, **wFUSION**, is used to aggregate several opinions about the same thing. The opinions being fused are weighted according to their importance or influence.
- Conflict-aware fusion, **cFUSION**, is additionally capable of dealing with the degree of conflict between opinions. When the difference of opinion becomes too significant, the certainty of the fused value drops.

2. Core API Documentation

This section presents the two classes implemented in the CertainTrust SDK.

- `CertainTrust` implements the trust calculation in both evidence representation and trust representation. It also implements all `CertainLogic` operators.
- `CertainTrustHTI` implements the visual representation for trust values, the Human Trust Interface.

2.1. The `CertainTrust` Class

The `CertainTrust` class models an opinion. Both representations – evidence-based and Trust/Certainty-based – are available concurrently and can be used together. An expectation value can be calculated from the opinion, too.

A `CertainTrust` object can be assigned to a `CertainTrustHTI` object. In this case, the HTI always updates whenever the `CertainTrust` object changes. If the HTI is not set to read-only, the user can modify the assigned `CertainTrust` object.

Any other object can register as observer for a `CertainTrust` object and will be notified on any changes. Any amount of observers is possible but might not make sense in all use cases.

- `CertainTrust(N)`
Creates a new `CertainTrust` object with $(r,s) = (0,0)$, $(t,c,f) = (0.5,0,0.5)$ and sets the amount of expected evidence to `N`.
- `CertainTrust(r, s, N)`
Creates a new `CertainTrust` object with supplied amount of positive (`r`) and negative (`s`) evidences and sets the amount of expected evidence to `N`.
- `CertainTrust(t, c, f, N)`
Creates a new `CertainTrust` object with supplied trust (`t`), certainty (`c`) and initial trust (`f`) values along with the amount of expected evidence `N`.
- `CertainTrust(t, c, f, N, DoC)`
Exactly like `CertainTrust(t, c, f, N)` but also sets the degree of conflict to `DoC`, which will be used by the following call of the `cFUSION` operator instead of the default value of 0.

Getters and Setters for all parameters are available and called `getX()`, `setX()`. Note that some parameters (like `t` and `c`) are only set together.

Method name	Functionality
<code>getC()</code>	Returns the certainty value.
<code>getT()</code>	Returns the trust value.
<code>setTC(newT, newC)</code>	Sets both certainty and trust.
<code>getF()</code>	Returns the initial trust value.
<code>setF(newF)</code>	Sets the initial trust value.
<code>getR()</code>	Returns the amount of positive evidence (can be float).
<code>addR(incR)</code>	Increases the amount of positive evidence by <code>incR</code> .
<code>getS()</code>	Returns the amount of negative evidence (can be float).
<code>addS(incS)</code>	Increases the amount of positive evidence by <code>incR</code> .
<code>setRS(newR, newS)</code>	Overrides the amount of both positive and negative evidence.
<code>getN()</code>	Returns the maximal amount of expected evidence.
<code>setN(newN)</code>	Sets the maximal amount of expected evidence.
<code>getDoC()</code>	Returns the degree of conflict. Only set by the <code>cFUSION</code> operator.
<code>setDoC(newDoC)</code>	Sets the degree of conflict.
<code>getExpectation()</code>	Returns the expectation assigned to the opinion.

Observers can be registered by calling `addObserver(newObserver)` and unregistered by calling `removeObserver(oldObserver)`. When a `CertainTrust` object changes, all observers' `update()` methods are called.

2.2. CertainLogic Operators

All `CertainLogic` Operators are implemented within the `CertainTrust` class as n-ary functions (except `NOT`). They are used in the following form:

```
// resultOpinion = ctObject1 OR ctObject2 OR ctObject3 OR ... OR ctObjectN
resultOpinion = ctObject1.OR(ctObject2, ctObject3, ..., ctObjectN);
```

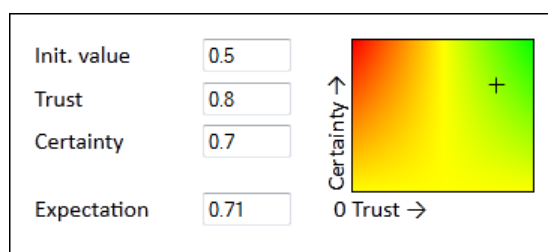
Observers are not copied to the newly created `resultOpinion` object.

In both Java and JavaScript, any amount (larger than zero) of opinions (as `CertainTrust` objects) can be supplied to the operator functions.

Method name	Functionality
<code>OR(opinion1, ...)</code>	Returns the result of <code>CertainLogic OR</code> applied to all supplied opinions.
<code>AND(opinion1, ...)</code>	Returns the result of <code>CertainLogic AND</code> applied to all supplied opinions.
<code>NOT(opinion)</code>	Returns the result of <code>CertainLogic NOT</code> applied to the opinion.
<code>wFusion([opinion1, ...], [weight1, ...])</code>	Returns the result of <code>CertainLogic wFUSION</code> (weighted fusion) applied to all supplied opinions with <code>weight1</code> for <code>opinion1</code> and so on.
<code>cFusion([opinion1, ...], [weight1, ...])</code>	Returns the result of <code>CertainLogic cFUSION</code> (conflict-aware fusion) applied to all supplied opinions with <code>weight1</code> for <code>opinion1</code> and so on. The returned object has the degree of conflict set.

2.3. The `CertainTrustHTI` Class

The `CertainTrustHTI` class implements a graphical representation of an opinion, intended for human usage. The graphical representation is called Human Trust Interface, thus HTI.



Objects of this class are always bound to a `CertainTrust` object which holds the opinion data to visualize. It is automatically registers as an observer for the `CertainTrust` object it is bound to. The user interface can be set to be read-only. Otherwise, a user can manipulate the associated `CertainTrust` object via the `CertainTrustHTI` interactively, but in case this is not wanted the interface can also be set to read-only to be purely for visualization.

- `CertainTrustHTI(certainTrust)`

Creates a `CertainTrustHTI` object and associates it with the supplied opinion `certainTrust`.

- `CertainTrustHTI(certainTrust, config)`
Same as `CertainTrustHTI(certainTrust)`, but additional configuration (described below) can be supplied. In JavaScript the parameter `config` has the type `object`, in Java it has the type `Map<String, String>`. All settings can be set independently and if not set their default values are used.

It is for example possible to set the HTI to be read-only in JavaScript with:

```
var hti = new CertainTrustHTI(myOpinion, {readonly: true});
```

In Java, the same can be achieved with:

```
Map<String,String> config = new HashMap<String, String>();  
config.put("readonly", "true");  
  
CertainTrustHTI hti = new CertainTrustHTI(myOpinion, config);
```

Option name	Functionality
<code>canvas.height</code>	Defines the HTI's height in pixels.
<code>canvas.width</code>	Defines the HTI's width in pixels.
<code>label.lang</code>	Use English (" <i>en</i> ") or German (" <i>de</i> ") localization as default for the labels. Detailed localization can be performed by assigning strings to <code>label.f</code> , <code>label.t</code> , <code>label.c</code> , and <code>label.e</code> while setting <code>label.lang</code> to any other value than " <i>en</i> " or " <i>de</i> ".
<code>label.f</code>	Sets the localization for the initial trust label.
<code>label.t</code>	Sets the localization for the trust label.
<code>label.c</code>	Sets the localization for the certainty label.
<code>label.e</code>	Sets the localization for the expectation label.
<code>readonly</code>	Disables any user input to the HTI. Specific elements can be set to read-only with <code>readonly.f</code> , <code>readonly.t</code> , <code>readonly.c</code> and <code>readonly.e</code> respectively. Modification with the mouse can be prevented with <code>readonly.mouse</code> .
<code>readonly.f</code>	Disables user input to the initial trust field.
<code>readonly.t</code>	Disables user input to the trust field.
<code>readonly.c</code>	Disables user input to the certainty field.
<code>readonly.e</code>	Disables user input to the expectation field.

readonly.mouse	Disables mouse input.
id <i>JavaScript only</i>	Assigns the HTML-id "certaintrust-widget-"+id to the created main DOM element.
line.cap <i>JavaScript only</i>	Defines the visuals for the crosshair: edge style.
line.height <i>JavaScript only</i>	Defines the visuals for the crosshair: line height.
line.width <i>JavaScript only</i>	Defines the visuals for the crosshair: line width.
line.style <i>JavaScript only</i>	Defines the visuals for the crosshair: line style.
domReturn <i>JavaScript only</i>	If set to true, the constructor does not return the HTI but its main element in the DOM.
domParent <i>JavaScript only</i>	Sets the parent DOM element under which the HTI is created as a child node. Used to control where the HTI shows up on an HTML page.
domBefore <i>JavaScript only</i>	Similar to domParent, but inserts the HTI before the supplied DOM element.
domAfter <i>JavaScript only</i>	Similar to domBefore, but inserts the HTI after the supplied DOM element.

To refresh a `CertainTrustHTI`'s UI even if the associated `CertainTrust` opinion did not change, call `update()`.

3. Using Evidences in CertainTrust

CertainTrust opinions are based on gathered evidences, which are binary experiences. Such evidence is either positive or negative. The ratio between positive (r) and negative (s) evidences defines the trust value of an opinion. The amount of evidences relates to the certainty value of an opinion. The expectation value is calculated using a Beta probability density function parameterized with r and s . See [Ries2009]¹ for details.



The amount of evidences in the CertainTrust API can be manipulated using the following three functions:

- **addR(x)**
This function adds x positive evidences to a CertainTrust object.
- **addS(x)**
Analogously, this function adds x negative evidences to a CertainTrust object.
- **setN(x)**
This function changes the expected amount of evidences.
- **setRS(r, s)**
This function replaces the current amount of evidences with r new positive evidences and s new negative evidences.

All three parameters can also be set using the CertainTrust constructor. Getter functions `getR()`, `getS()`, and `getN()` are available.

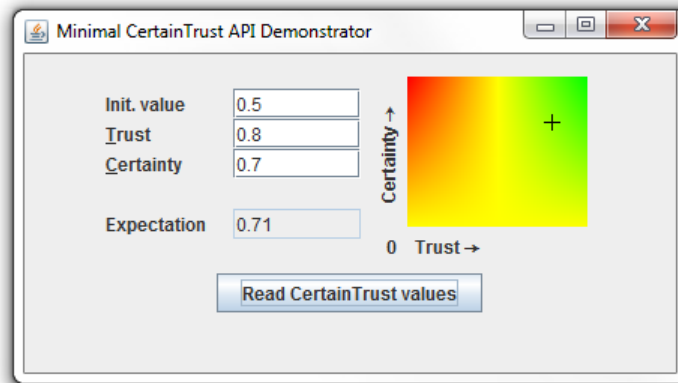
The JavaScript demonstrator **evidences.html** provides a CertainTrust object and the possibility to manipulate the amount of collected evidences.

¹ Sebastian Ries. Extending Bayesian Trust Models Regarding Context-Dependence and User Friendly Representation. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*, ACM Press, 2009. Download available at: <http://dl.acm.org/citation.cfm?id=1529573>

4. Step-By-Step Guide: Java

We expect you to be familiar with your development environment of choice and the Java programming language (including Swing) in general.

In the following, you will build a small demonstrational Java application that displays a `CertainTrust` object visually and accesses its attributes programmatically. The user interface looks similar to the screenshot below.



You can find the complete source code for this example in the folder **Documentation/JavaDemonstrator** and the file **Minimal.java**.

4.1. Importing the CertainTrust SDK JAR

As a first step after creating your project, you must import the `CertainTrust` SDK JAR file which is usually named **CertainTrustSDK.jar**, maybe including a version number in the filename. To accomplish this in the Eclipse Development Environment, copy the file into any of your project's (sub-)directories, right-click the file in the Package Explorer and select *Build Path -> Add to Build Path*.

Now, the namespace `CertainTrust` should be available in your project. The two relevant classes in this namespace are

- `CertainTrust.CertainTrust`, which holds the data objects encapsulating the (c,t,f)-triples and the methods to apply `CertainLogic` operators to other `CertainTrust` objects
- `CertainTrust.CertainTrustHTI`, which is the Swing-based user interface for displaying and manipulating `CertainTrust` objects.

4.2. Project Skeleton Code

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Minimal extends JFrame implements ActionListener {
    private static final long serialVersionUID = -447167281994322634L;

    // this object stores the trust data and implements the operators
    CertainTrust ctObject;

    public Minimal() {
        setTitle("Minimal CertainTrust API Demonstrator");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // CertainTrust goes here

        // add a button to read the CertainTrust values
        JButton button = new JButton("Read CertainTrust values");
        add(button);

        this.setSize(450, 250);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new Minimal();
    }
}
```

This code creates a single window with a button.

4.3. Creating and Displaying a Human Trust Interface

To display a single HTI, a `CertainTrust` data object that stores the (c,t,f)-triple is needed. A `CertainTrust` data object is created by calling its constructor:

```
import CertainTrust.CertainTrust;

...

CertainTrust ctObject = new CertainTrust(10);
```

This constructor takes one parameter: the value of N , the maximal expected evidence. Other constructors are available, please refer to prior sections.

When a data object is available, an HTI can be created:

```
import CertainTrust.CertainTrust;
import CertainTrust.CertainTrustHTI;

...

// this object stores the trust data and implements the operators
CertainTrust ctObject;

...

// display a single HTI
ctObject = new CertainTrust(10);
add(new CertainTrustHTI(ctObject));
```

After this step, the user interface is final and should look similar to the one in the screenshot.

Each HTI is automatically bound to its data object via the observer pattern. This means that every change made in the HTI automatically propagates back to the data object.

4.4. Programmatically Accessing CertainTrust Data

The demonstrator includes a button underneath the HTI. When the button is clicked, a popup window should appear and show the (c,t,f)-triple stored in the CertainTrust data object.

For the sake of simplicity in this example, the Minimal class used for the JFrame also serves as ActionListener for the button.

```
@Override
public void actionPerformed(ActionEvent arg0) {
    // whenever the button is clicked
    JOptionPane.showMessageDialog(this,
        "Values of the CertainTrust object:\n"
        + "\nInit. value: " + this.ctObject.getF()
        + "\nTrust: " + this.ctObject.getT()
        + "\nCertainty: " + this.ctObject.getC()
        + "\nExpectation: " + this.ctObject.getExpectation());
}
```

All properties of the CertainTrust data object can be accessed using the appropriate getter functions. Although in our example, the data object is manipulated using the HTI, any property can also be set programmatically by calling CertainTrust's setters.

The complete source code of the final example is printed below. Try it yourself!


```

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

import CertainTrust.CertainTrust;
import CertainTrust.CertainTrustHTTI;

public class Minimal extends JFrame implements ActionListener {
    private static final long serialVersionUID = -447167281994322634L;

    // this object stores the trust data and implements the operators
    CertainTrust ctObject;

    public Minimal() {
        setTitle("Minimal CertainTrust API Demonstrator");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // display a single HTI
        ctObject = new CertainTrust(10); // first, we need a CertainTrust data object
        add(new CertainTrustHTTI(ctObject));

        // add a button to read the CertainTrust values
        JButton button = new JButton("Read CertainTrust values");
        button.addActionListener(this);
        add(button);

        this.setSize(450, 250);
        this.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent arg0) {
        // whenever the button is clicked
        JOptionPane.showMessageDialog(this,
            "Values of the CertainTrust object:\n"
            + "\nInit. value: " + this.ctObject.getF()
            + "\nTrust: " + this.ctObject.getT()
            + "\nCertainty: " + this.ctObject.getC()
            + "\nExpectation: " + this.ctObject.getExpectation());
    }

    public static void main(String[] args) {
        new Minimal();
    }
}

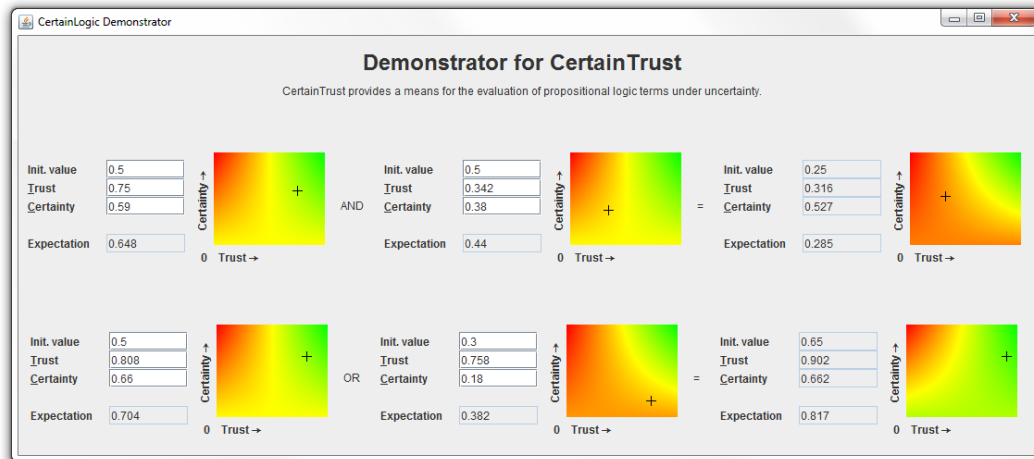
```

4.5. Using CertainLogic

In this tutorial, the task is to implement a full demonstrator for the CertainLogic operators **AND** and **OR**. It requires the knowledge of the above tutorial.

The source code is available in [Documentation/JavaDemonstrator/demonstrator.java](#).

The Java applet allows the users to calculate an **AND** as well as an **OR** operation of two CertainTrust data objects:



At the start, six `CertainTrust` data objects are created and bound to HTIs. The four HTIs on the left serve as input parameters for the operations, the two ones on the right display the results and are therefore set to be read-only.

This setting can be made at the creation of the HTI by supplying a `Map` that contains the configuration options as `String` pairs. Please refer to section 2.3 for all available options.

```
Map<String,String> htiConfig = new HashMap<String, String>();
htiConfig.put("readonly", "true");

new CertainTrustHTI(new CertainTrust(10), htiConfig);
```

To keep the rightmost HTIs updated, an `Observer` is bound to each of the two HTIs serving as input elements for the operands.

In the demonstrator, a new class called `ANDObserver` is implemented to calculate the `AND` value of the first two `CertainTrust` data objects and to display the result in the rightmost HTI.

```

import java.util.Observable;
import java.util.Observer;
import CertainTrust.CertainTrust;

public class ANDObserver implements Observer {

    private CertainTrust Operand1;
    private CertainTrust Operand2;
    private CertainTrust Result;

    public ANDObserver(CertainTrust Operand1, CertainTrust Operand2, CertainTrust Result) {
        this.Operand1 = Operand1;
        this.Operand2 = Operand2;
        this.Result = Result;
    }

    @Override
    public void update(Observable arg0, Object arg1) {
        // calculate the result of the AND operation
        // and update the Result CertainTrust data object
        CertainTrust ANDResult = this.Operand1.AND(this.Operand2);
        this.Result.setF(ANDResult.getF());
        this.Result.setTC(ANDResult.getT(), ANDResult.getC());
    }
}

```

An instance of this class is registered to observe the input HTIs.

```

// wire all components forming the AND operation
ANDObserver andObserver = new ANDObserver(AndOperand1, AndOperand2, AndResult);
AndOperand1.addObserver(andObserver);
AndOperand2.addObserver(andObserver);

```

The most important line in the code of ANDObserver is the application of the **AND** operator to the operands:

```

CertainTrust ANDResult = this.Operand1.AND(this.Operand2);

```

The **AND()** method of **CertainTrust** creates a new **CertainTrust** data object holding the result of the calculation. The outcome of the calculation done is *Operand1 AND Operand2*. Note that the first operand is always the data object, from which the **AND()** method is called.

Furthermore, the method consumes any amount of parameters. This allows calculations like *Operand1 AND Operand2 AND Operand3 AND ... OperandN*.

```

CertainTrust ANDResult = this.Operand1.AND(this.Operand2, this.Operand3, this.Operand4, this.Operand5);

```

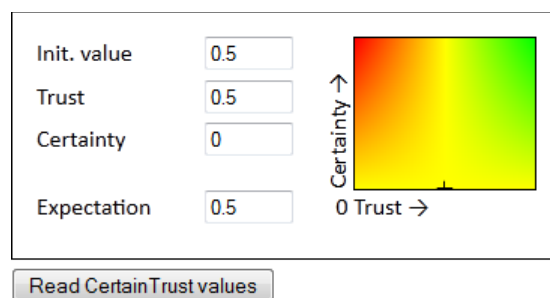
The same process is used to “wire up” the **OR** operation.

5. Step-By-Step Guide: JavaScript

We expect you to be familiar with your development environment of choice, JavaScript programming and basic HTML.

The HTI requires the HTML5 canvas object to draw the user interface. In older versions of HTML, the SDK is not available and/or produces unwanted results (depending on the browser).

In the following, you will build a small demonstrational web page with the same function that can be found in the Step-By-Step tutorial for the Java language. The web page will look similar to the screenshot below:



You can find the complete source code for this example in the folder **Documentation/JavaScriptDemonstrator** and the file **minimal.html**.

5.1. Project Skeleton Code

For any HTML5 page, two script files and one stylesheets need to be imported: **CertainTrust.js** contains the **CertainLogic** data object and operators, while **CertainTrustHTI.js** implements the visual interface. The **CertainTrustHTI.css** file includes standard styles for the visual interface. To change the visual representation, just edit the CSS file.

Simply copy the three files into the same directory as your HTML file or adapt the path definitions of the script and link tags to your needs.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CertainTrust Demonstrator in JavaScript</title>

  <!-- include these two scripts and the CSS to enable both CertainTrust and the HTI -->
  <script type="text/javascript" src="CertainTrust.js"></script>
  <script type="text/javascript" src="certainTrustHTI.js"></script>
  <link rel="stylesheet" type="text/css" href="certainTrustHTI.css"/>
</head>
<body>
  <script type="text/javascript">
    // CertainTrust goes here
  </script>
</body>
</html>
```

5.2. Creating and Displaying a Human Trust Interface

To display a single HTI, a `CertainTrust` data object that stores the (c,t,f)-triple is needed. A `CertainTrust` data object is created by creating a new instance. The parameter is `N`, the maximal expected evidence. More parameters are available, please refer to prior sections.

```
var ctObject = new CertainTrust(10);
```

When a data object is available, an HTI can be created. It is inserted into the DOM tree wherever the call with `new` is executed. More fine-grained control over the DOM tree position is possible. Please refer to section 2.3 for all options.

```
var ctObject = new CertainTrust(10);
var hti = new CertainTrustHTI(ctObject);
```

Each HTI is automatically bound to its data object via the observer pattern. This means that every change made in the HTI automatically propagates back to the data object.

5.3. Programmatically Accessing the CertainTrust Data

The demonstrator includes a button underneath the HTI. When the button is clicked, a popup window should appear and show the (c,t,f)-triple stored in the `CertainTrust` data object.

```

<script type="text/javascript">
  var ctObject = new CertainTrust(10);
  var hti = new CertainTrustHTI(ctObject);

  function showValues() {
    alert("Values of the CertainTrust object:\n"
      + "\nInit. value:\t" + ctObject.getF()
      + "\nTrust:\t" + ctObject.getT()
      + "\nCertainty:\t" + ctObject.getC()
      + "\nExpectation:\t" + ctObject.getExpectation());
  }
</script>
<br />
<button type="button" onclick="showValues();">Read CertainTrust values</button>

```

All properties of the `CertainTrust` data object can be accessed using the appropriate getter functions. Although in our example, the data object is manipulated using the HTI, any property can also be set programmatically by calling `CertainTrust`'s setters.

The complete source code of the final example is printed below. Try it yourself!

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CertainTrust Demonstrator in JavaScript</title>

  <!-- include these two scripts and the CSS to enable both CertainTrust and the HTI -->
  <script type="text/javascript" src="CertainTrust.js"></script>
  <script type="text/javascript" src="certainTrustHTI.js"></script>
  <link rel="stylesheet" type="text/css" href="certainTrustHTI.css"/>
</head>
<body>
  <script type="text/javascript">
    var ctObject = new CertainTrust(10);
    var hti = new CertainTrustHTI(ctObject);

    function showValues() {
      alert("Values of the CertainTrust object:\n"
        + "\nInit. value:\t" + ctObject.getF()
        + "\nTrust:\t" + ctObject.getT()
        + "\nCertainty:\t" + ctObject.getC()
        + "\nExpectation:\t" + ctObject.getExpectation());
    }
  </script>
  <br />
  <button type="button" onclick="showValues();">Read CertainTrust values</button>
</body>
</html>

```

5.4. Using CertainLogic

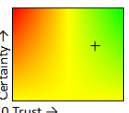
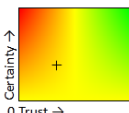
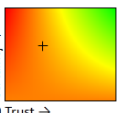
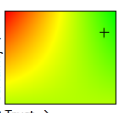
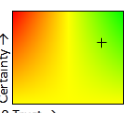
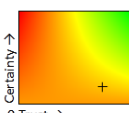
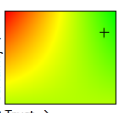
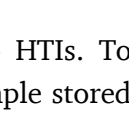
In this tutorial, the task is to implement a full demonstrator for the `CertainLogic` operators **AND** and **OR**. It requires the knowledge of the above tutorial.

The source code is available in [Documentation/JavaScriptDemonstrator/demonstrator.html](#).

The web page allows the users to calculate an **AND** as well as an **OR** operation of two CertainTrust data objects:

Demonstrator for CertainTrust

CertainTrust provides a means for the evaluation of propositional logic terms under uncertainty.

Init. value: 0.5 Trust: 0.75 Certainty: 0.59 Expectation: 0.648 Certainty →  + 0 Trust →	AND	Init. value: 0.5 Trust: 0.342 Certainty: 0.38 Expectation: 0.44 Certainty →  + 0 Trust →
Init. value: 0.25 Trust: 0.338 Certainty: 0.588 Expectation: 0.302 Certainty →  + 0 Trust →	=	Init. value: 0.25 Trust: 0.338 Certainty: 0.588 Expectation: 0.302 Certainty →  + 0 Trust →
Init. value: 0.5 Trust: 0.808 Certainty: 0.66 Expectation: 0.703 Certainty →  + 0 Trust →	OR	Init. value: 0.3 Trust: 0.758 Certainty: 0.18 Expectation: 0.382 Certainty →  + 0 Trust →
Init. value: 0.65 Trust: 0.901 Certainty: 0.767 Expectation: 0.842 Certainty →  + 0 Trust →	=	Init. value: 0.65 Trust: 0.901 Certainty: 0.767 Expectation: 0.842 Certainty →  + 0 Trust →

At the start, six CertainTrust data objects are created and bound to HTIs. To create an HTI as the child of a specific DOM tree element (in our example stored in the variable `DOMElement`), supply the parameter `domParent` to the constructor:

```
var ctObject = new CertainTrust(10);
var HTI = new CertainTrustHTI(ctObject, {domParent: DOMElement});
```

The four leftmost HTIs serve as input parameters for the operations, the two rightmost ones display the results and are therefore set to be read-only.

This setting can be made by supplying a `readonly` parameter to the constructor. Please refer to the documentation for all available options.

```
var ctObject = new CertainTrust(10);
var HTI = new CertainTrustHTI(ctObject, {readonly: true});
```

To keep the rightmost HTIs updated, an observer is bound to each of the two HTIs that serve as input elements for the operands.

In the demonstrator, a new prototype called `ANDObserver` is implemented to calculate the **AND** value of the first two CertainTrust data objects and to display the result in the rightmost HTI.

```
// ANDObserver is used for the AND calculation
var ANDObserver = {
  update: function() {
    // calculate the CertainTrust.AND for both values
    var CT_result = ctAndOperator1.AND(ctAndOperator2);

    // update the HTI which displays the result
    ctAndResult.setF(CT_result.getF());
    ctAndResult.setTC(CT_result.getT(), CT_result.getC());
  }
};
```

The ANDObserver is registered to observe the input HTIs.

```
ctAndOperator1.addObserver(ANDObserver);  
ctAndOperator2.addObserver(ANDObserver);
```

The most important line in the code of ANDObserver is the application of the AND operator to the operands:

```
var CT_result = ctAndOperator1.AND(ctAndOperator2);
```

The AND() method of CertainTrust returns a new CertainTrust data object instance holding the result of the calculation. The outcome of the calculation done is *Operand1 AND Operand2*. Note that the first operand is always the data object, from which the AND() method is called.

Furthermore, the method consumes any amount of parameters. This allows calculations like *Operand1 AND Operand2 AND Operand3 AND ... OperandN*.

```
var CT_result = ctAndOperator1.AND(ctAndOperator2, ctAndOperand3, ctAndOperand4, ctAndOperand5);
```

The same process is used to “wire up” the OR operation.

Acknowledgement

The work presented here was performed in the context of the Software-Cluster project InDiNet (www.software-cluster.org) and funded by the German Federal Ministry of Education and Research (BMBF) under grant no. "01IC10S04". The authors assume responsibility for the content.

The Source Code is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, you can obtain one at <http://mozilla.org/MPL/2.0/>.