# ConflictDetector User Guide

Version 1.0, October 2011

Corporate Headquarters
iPass Inc.
3800 Bridge Parkway
Redwood Shores, CA 94065 USA

www.ipass.com
+1 650-232-4100
+1 650-232-0227 fx

TABLE OF CONTENTS

TABLE OF CONTENTS

# 1 Introduction

## 1.1 What is ConflictDetector?

ConflictDetector is a general framework for detecting and resolving specific conflicts in the system.

This document applies ConflictDetector to resolve conflicts between the Open Mobile Connection Manager and other connections managers. Connection managers use various system resources (such as network adapters, networks, etc.) and some require exclusive use of these resources. This may interfere with Open Mobile operations, resulting in conflicts. Therefore, Open Mobile needs to detect and resolve such conflicts to function with these connection managers. Ideally, this should happen non-intrusively with maximum transparency to the user.

## 1.2 Configuring ConflictDetector

ConflictDetector configuration is defined in the `ConflictDetectionConfig.xml` file located in the Open Mobile Profile. This file can contains one or more Conflict Definitions.

Creating a Conflict Definition is an involved, manual process that requires a knowledgeable IT engineer. To simplify this process, iPass provides numerous Conflict Definitions. This includes the Conflict Definitions for the most commonly used connections managers, such as Cisco NAM and Funk/Juniper Odyssey. Multiple generic samples are also provided.

Those samples can be downloaded from the iPass Open Mobile Portal or they can be provided separately upon request.

## 1.3 User Interaction

The ConflictDetector can switch between connections managers (according to network visibility, location, and other factors) without the user's awareness. However, the administrator may configure ConflictDetector to interact with the user:

- To allow users to manually resolve conflicts
- To display alert messages when conflicts are resolved

# 2 ConflictDetector Basics

The ConflictDetector's basic functions can be explained using the simple example of a motion sensor. The function of a motion sensor is to turn on the light when it is dark and when movement is detected. This example contains all of the elements necessary to describe basic ConflictDetector functionality and terms.

## 2.1 Basic Terms

The following list of terms are followed by their equivalent in the motion sensor example

### 2.1.1 Major Terms

- **Conflict Title:** The room is dark.
- **Applicability:** The motion sensor is installed.
- **Condition:** Is the room dark?
- **Resolve Conflict Trigger:** On motion detected.
- **Resolve Conflict Actions:** Turn the light on.
- **Restore State Trigger:** Motion no longer detected.
- **Restore State Actions:** Turn the light off.

### 2.1.2 Conflict States

- **Conflict Absent:** The room is lit because there is daylight or someone manually turned on a light. Since the motion sensor did not resolve the conflict, it cannot restore the state (to darkness).
- **Conflict Present:** The room is dark.
- **Actively Enforced:** Someone enters the dark room triggering the motion sensor, and the motion sensor resolves the conflict by turning on the light. We call this **Enforcement**.
- **Failure to Resolve the Conflict:** In case the light bulb is burnt-out or absent, Motion Sensor turns the switch on, but the darkness condition is still present. Therefore, the Motion Sensor detects that it **failed** to resolve the condition.

The following table shows these states (as depicted in the ConflictMonitor):

| Conflict State | Conflict Depiction | Enforced |
|---|---|---|
| **Absent** | ~~The room is dark~~ | |
| **Present** | The room is dark | |
| **Resolved** | ~~The room is dark~~ | **V** |
| **Failed** | The room is dark | |

## 2.2 Basic ConflictDetector Operations

There are two basic ConflictDetector operations:

- Trigger Processing
- Conflict Monitoring

### 2.2.1 Trigger Processing

The ConflictDetector processes each Conflict Definition by performing the following actions:

1. **Checks Applicability**. If the condition is inapplicable (if the motion sensor is not installed), it skips the conflict processing.

2. When the conflict is in a **non-enforced state** (the lights are off), on **Resolve Conflict Trigger** (the motion is detected) the **Condition** (is the room dark?) is checked. If the condition is true (it is dark), the **Resolve Actions** are performed (the lights are turned on). Then the condition is checked again.
    - If the condition is evaluated as false (it is no longer dark) the conflict is considered **Actively Enforced**.
    - If the condition is evaluated as true (it is still dark) the conflict is considered **Failed**.
3. When the conflict is in an **enforced state** (the motion sensor turned on the lights), on **Restore State Trigger** (no motion is detected):
    - The **Condition** is checked (is it dark?).
    - If the condition is false (the room is lit), the **Restore Actions** are performed (the lights are turned off).
    - The conflict state becomes **non-enforced**.

### 2.2.2 Conflict Monitoring

The ConflictDetector is actively trying to enforce the right state. However, conditions may change beyond the ConflictDetector's control. Using the motion sensor as an example:

- When the room is dark, someone turns on a light.
- After the condition is enforced (the motion sensor turns on the light), someone turns off the light.
- The light bulb fails.

ConflictDetector monitors conflict conditions on a regular time interval. If the conflict condition is changed, the conflict state is properly updated.

Monitoring PollInterval is defined in `ConflictDetectionConfig.xml`. A typical time interval is about 10 seconds.

# 3 ConflictDetector XML Configuration

This chapter introduces ConflictDetector configuration, which is expressed in XML. This configuration is general and could be applied to multiple domains.

## 3.1 Top-Level Structure

ConflictDetectionConfig.xml has two main sections: Resources and Conflicts.

```
<ConflictDetectionConfig>
   <Resources>
   </Resources>
   <Conflicts>
+     <Conflict>
+     <Conflict>
   </Conflicts>
</ConflictDetectionConfig>
```

**1. Resources**: defines resources that are used in the Conflicts section.

**2. Conflicts**: defines conflicts, one at a time,

## 3.2 Resources Element

The Resources element contains two sections: Variable and Actions.

```
   <Resources>
      <Variables>
+        <int.var.def id="intVar1">
+        <bool.def id="boolVar2">
+        <int.var.def id="intVar3">
      </Variables>
      <Actions>
+        <action.def id="action1">
+        <action.def id="action2">
    </Actions>
   </Resources>
```

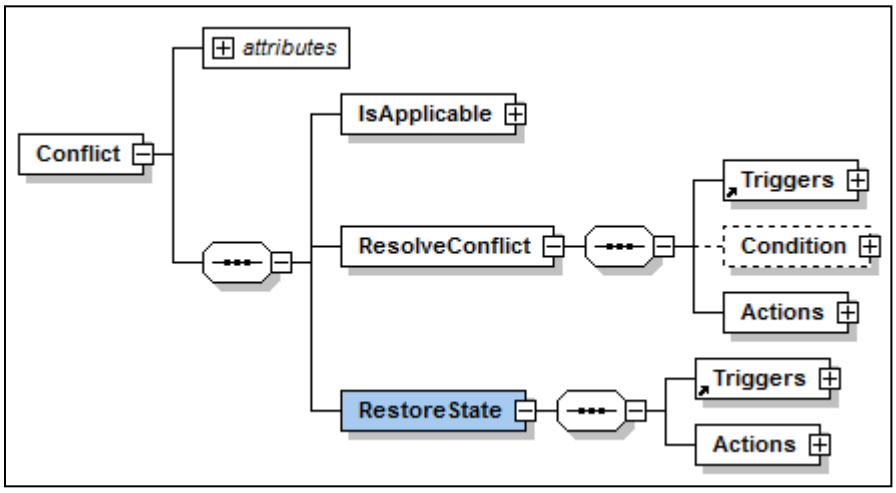
**1. Variables**: defines variables.

**2. Actions**: defines actions.

Variable and Actions definitions are described in the Expressions section. Below is an example that defines the Boolean function IsWiFiEnabled and Action EnableWiFi.

```
<Resources>
   <Variables>
      <bool.def id="IsWiFiEnabled">
         <IsMediaManaged MediaType="WiFi"/>
      </bool.def>
   </Variables>
   <Actions>
      <action.def id="EnableWiFi">
         <ManageMedia MediaType="WiFi"/>
         <Sleep Time="00:00:10"/>
      </action.def>
   </Actions>
</Resources>
```

## 3.3 Conflict Element




Each Conflict element defines a separate conflict in the system. Below is an example:

```
<Conflict id="NotepadCalcExclusion" Title="Allow Notepad only if Calc is not running">
  <IsApplicable>
  <ResolveConflict>
+     <Triggers>
+     <Condition>
+     <Actions>
  </ResolveConflict>
  <RestoreState>
+     <Triggers>
+     <Actions>
  </RestoreState>
</Conflict>
```

The Conflict element consists of the following:

| XML Tag | Description |
|---|---|
| <Conflict id="[id]" Title="[Title]"> | This header contains id and Title. id is used to refer to this conflict programmatically. Title is used to display the conflict in the ConflictMonitor User Interface. |
| <IsApplicable> | Defines whether this conflict exists in the system. For example, whether a particular program is installed or a service is defined. |
| <ResolveConflict> | Defines how to detect and resolve a conflict. It contains the following elements:<br>• <Triggers> that defines when to check for the conflict<br>• <Condition> that defines the condition under which the conflict is present;<br>• <Actions> that defines the actions that will resolve the conflict. |
| <RestoreState> | Used to define when and how to restore the original state (or reverse the previous actions). It contains the following elements:<br>• <Triggers> that define when to restore the state (only occurs if the <Condition> is false)<br>• <Actions> that defines actions that will restore the state. |

# 4 XML Vocabulary

ConflictDetector framework is a general framework that applies to multiple technology domains. Adapting it to a specific domain requires a domain-specific vocabulary. This chapter defines ***Connection Manager Vocabulary*** that allows ConflictDetector to resolve conflicts between Open Mobile and other connection managers.

The vocabulary is divided into:

- Trigger Vocabulary
- Expressions Vocabulary
- Action Vocabulary

## 4.1 Conventions

The vocabulary entries are described in a tabular form. For example:

| Function | Attributes | Description |
|---|---|---|
| IsConnectionStateEq | opt MediaType<br>req ConnectionState | Checks if the current connection state for MediaType is equal to a specified ConnectionState (Connected, Warm, Disconnected). |

**Function**: (IsConnectionStateEq) defines its name.

**Attributes**: The function may have attributes (MediaType, ConnectionState). Optional attributes are marked with an opt prefix. If a parameter is not marked as optional, it is required. For clarity and symmetry some required parameters are marked with a req prefix.

## 4.2 Trigger Vocabulary

Trigger vocabulary defines various trigger points where the ConflictDetector checks the state in order to perform certain actions.

| Trigger Name | Attributes | Description |
|---|---|---|
| OnUserRequest | Message | This defines a user trigger. Message is used as a trigger caption (for example, displayed on a trigger button). The trigger is triggered when the user requests the action (for example, by pushing the button). |
| | | |
| OnCondition | | Requires a condition (Boolean expression) as a child element. Triggered when the condition is evaluated to true. |
| OnConditionRef | idref | Similar to OnCondition, but the condition is provided by idref. Triggered when the condition is evaluated to true. |
| | | |
| OnServiceStart | | Triggered when OM starts (on service start). |
| OnServiceStop | | Triggered just before OM stops (on service shutdown). |
| OnWakeup | | Triggered on waking up from sleep or hibernation. |
| | | |
| OnLaunchingUI | | Triggered when OM UI is launched. |
| OnExitingUI | | Triggered when OM UI exits. |
| | | |
| OnInternetPreconnect | | Triggered just before the Internet connection starts. |
| OnInternetDisconnected | | Triggered after the Internet gets disconnected. |

## 4.3 Expression Vocabulary

Expressions here are typical mathematical expressions. As a typical mathematical expression, an expression consists of variables and operators applied to the variables. Variable is defined using a ***predefined vocabulary***.

An expression is evaluated into a value. The resulting value type determines the ***expression type***. Currently, we support the following expression types:

- Boolean (or logical) expressions
- Integer expressions
- String expressions

Other expression types may be added in the future, as needed.

A vocabulary is subdivided into the following vocabulary groups:

- Boolean vocabulary
- Integer vocabulary
- Version vocabulary

### 4.3.1 Boolean Vocabulary

Boolean vocabulary defines functions returning Boolean values:

| Function | Attributes | Description |
|---|---|---|
| **IsMediaManaged** | MediaType | Checks if the specified media (Wi-Fi, Ethernet, Mobile Broadband) is managed by Open Mobile. |
| | | |
| **IsConnectionStateEq** | opt MediaType<br>req ConnectionState | Checks if the current connection state for MediaType is equal to a specified ConnectionState (Connected, Warm, Disconnected).<br>▪ Warm means non-disconnected.<br>▪ If MediaType is omitted, the function checks if any media has a specified ConnectionState. |
| | | |
| **IsDirExist** | DirName | Checks if the specified dir exists |
| **IsFileExist** | FileName | Checks if the specified file exists |
| | | |
| **IsProcessRuning** | ProcessName | Checks if the specified process is running |
| **IsServiceExist** | ServiceName | Checks if the specified service exists |
| **IsServicePaused** | ServiceName | Checks if the specified service is paused |
| **IsServiceRunning** | ServiceName | Checks if the specified service is running |
| **IsNetworkPresent** | opt MediaType<br>opt NetworkName<br>opt Is8021x<br>opt IsNonBroadcast<br>opt ConnectionState | Checks if at least one network with the defined attributes is present. All the attributes are optional. If an attribute is missing, it is irrelevant for matching:<br>▪ If none of the attributes are specified, it is triggered if any network is present<br>▪ If only MediaType is specified, it is triggered if a network with this MediaType is present<br>▪ If only NetworkName is specified, it is triggered if a network with this NetworkName is present<br>▪ If only Is8021x is specified, it is triggered if any 8021x network is present<br>▪ If only IsNonBroadcast is specified, it is triggered if any non-broadcast network is present<br>▪ If NetworkName and Is8021x are specified, it is triggered if a network with this NetworkName is present and it is an 8021x |
| **IsConnectedTo--CorporateNetwork** | opt CNDName | Checks if connected to a corporate network. |
| | | |
| **IsRegistryPresent** | req Key<br>opt ValueName | Checks if the specified registry is present. |
| **IsRegistryEq** | req Key<br>req ValueName<br>req To | Checks if the specified registry value is equal to the specified To value. |

### 4.3.2 Integer Vocabulary

Integer vocabulary defines functions returning Integer values.

| Function | Attributes | Description |
|---|---|---|
| **OSVersion.Major** | | Returns OS Major Version number |
| **OSVersion.Minor** | | Returns OS Minor Version number |

### 4.3.3 Version Vocabulary

Version vocabulary defines functions returning version values. A version is defined as a string. A typical version string is represented in “int.int.int” format. For example: “3.436.25”. For more details see the Version Operators below.

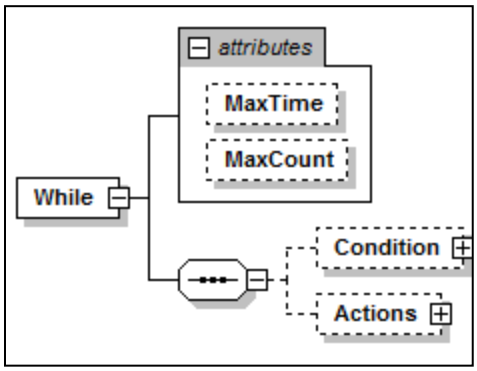
| Function | Attributes | Description |
|---|---|---|
| **FileVersion** | FileName | Returns file version. Typically used with .exe and .dll files. |

## 4.4 Actions

The following Actions are defined:

| Action | Description |
|---|---|
| **Sleep**(Time) | Sleeps for the specified time |
| | |
| **InformUser**(Messsage) | Displays a specified Message to the user |
| **ReprobeNetwork**(MediaType, NetworkName) | Refreshes a network |
| **ReprobeNetworks**(MediaType)<br>{<br>Network(NetworkName)<br>…<br>Network(NetworkName)<br>} | Refreshes a list of networks |
| | |
| **ManageMedia**(MediaType) | Allows Open Mobile to manage the specified media |
| **UnmanageMedia**(MediaType) | Disallows Open Mobile to manage the specified media |
| | |
| **StartProcess**(opt Context , ProcessName, opt Args) | Starts the process in the specified context. Default is the User context. Arguments to the process are provided in Args. |
| **StopProcess**(ProcessName) | Stops the specified process |
| **StartService**(ServiceName) | Starts the specified service |
| **PauseService**(ServiceName) | Pauses the specified service |
| **ResumeService**(ServiceName) | Resumes the specified service |
| **StopService**(ServiceName) | Stops the specified service |
| | |
| **SetRegistry**(Key, ValueName, Value) | Sets the specified registry. |
| **RemoveRegistry**(Key) | Removes the specified registry. |
| | |
| **While**(opt MaxTime, opt MaxCount)<br>{<br>opt Condition<br>opt Actions<br>} | Executes multiple actions in the while-loop. The loop exists when MaxCount is exhausted, MaxTime is exhausted, or Condition fails. |
| | |
| **action**.ref(idref) | Executes a custom action (more details below). |

### 4.4.1 While Action



While-action allows the execution of multiple actions in the loop. It loops until optional MaxCount is exhausted, optional MaxTime is exhausted, or Condition fails.

Its main purpose is to wait until some external condition is met. For example, after starting a third-party connection manager it waits up to 20 sec until the third party user interface is displayed (as shown in the following XML):

```
<Actions>
   <StartProcess ProcessName="nam.exe"/>
   <While MaxTime="00:00:20">
      <Condition>
         <not><IsProcessRunning ProcessName="namgui.exe"/></not>
      </Condition>
      <Actions>
         <Sleep Time="00:00:20"/>
      </Actions>
   </While>
</Actions>
```

> *While-action does not have any default timeouts and must be used very carefully.*

### 4.4.2 Custom Actions

The framework allows for the creation of a custom action, an action that contains or refers to other actions. For example:

```
<action.def id="EnableWiFi">
   <ManageMedia MediaType="WiFi"/>
   <Sleep Time="00:00:10"/>
</action.def>
```

# 5 XML Expressions

XML expressions are typical mathematical expressions. As a typical mathematical expression it consists of variables and operations applied to the variables.

An expression is evaluated into a value. The resulting value type determines the ***expression type***. Currently we support the following expression types:

- Boolean (or logical) expressions
- Integer expressions
- String expressions

## 5.1 Operators

Operators are divided into the following groups:

- Boolean operators
- Integer operators
- Version operators

### 5.1.1 Boolean Operators

Boolean operators are conventional operators returning Boolean values.

| Operator | Attributes | Description |
|---|---|---|
| **bool.const** | value | Defines Boolean constant (true or false). |
| | | |
| **not** | | Logical not operator. |
| **and** | | Logical and operator. Two or more Boolean operands are expected. |
| **or** | | Logical or operator. Two or more Boolean operands are expected. |
| | | |
| **not.ref** | idref | Equivalent to not operator, except the Boolean condition is provided by the reference. |
| | | |
| **int.neq**<br>**int.lt**<br>**int.leq**<br>**int.eq**<br>**int.geq**<br>**int.gt** | | Compare 2 integers for non-equality, less, less-or-equal, equal, greater-or-equal, greater respectively. |
| | | |
| **string.eq** | | Checks if two strings are equal |

### 5.1.2 Integer Operators

Integer operators are conventional mathematical operators returning integer values.

| Operator | Attributes | Description |
|---|---|---|
| **int.const** | value | Defines integer constant |
| | | |
| **int.add**<br>**int.sub**<br>**int.mul**<br>**int.div**<br>**int.mod** | | Conventional integer operators |

### 5.1.3 Version Operators

Version operators are non-conventional operators comparing versions (as returned by FileVersion).

| Operator | Attributes | Description |
|---|---|---|
| **version.neq**<br>**version.lt**<br>**version.leq**<br>**version.eq**<br>**version.geq**<br>**version.gt** | | Compare a version against a pattern for non-equality, less, less-or-equal, equal, greater-or-equal, greater respectively. |

Using this comparison:

| | | |
|---|---|---|
| “1.1” | equal | “1.1” |
| “1.1” | equal | “1.x” |
| “1.2.foo” | equal | “1.x” |
| “1.1” | greater | “1.0” |
| “1.1” | greater | “1.0.999” |
| “1.1” | greater | “1.0.x” |

## 5.2 Custom Variables

Custom variables can be defined. A custom variable represents an expression. It could be a simple expression, such as a constant or a reference to a predefined vocabulary function. It could be a complex expression, which may contain operations and other custom variables. Currently, we support Boolean and Integer variables. A variable is identified by id and is defined by providing an expression body. For example:

```
<bool.def id="IsWiFiEnabled">
  <IsMediaManaged MediaType="WiFi"/>
</bool.def>

<bool.def id="IsMBEnabled">
  <IsMediaManaged MediaType="MB"/>
</bool.def>
```

The custom variables can be used in conditions. They can also be used to create more complex variables. For example:

```
<bool.def id="IsMediaManaged">
  <or>
    <bool.ref idref="IsWiFiEnabled"/>
    <bool.ref idref="IsMBEnabled"/>
  </or>
</bool.def>
```

| Operator | Attributes | Description |
|---|---|---|
| **bool.def** | id | Defines Boolean variable |
| **int.def** | id | Defines Integer variable |
| **bool.ref** | idref | Reference to a Boolean variable |
| **int.ref** | idref | Reference to an Integer variable |

Attribute ‘id’ specifies a variable name. Its XSD type is “xs:ID”. It imposes some beneficial restrictions on valid names:

1. The name should be syntactically correct. It must begin with a letter and cannot contain empty spaces or special characters. The valid names are Foo, Bar, Foo1234, foo_bar, foo.bar, foo-bar. The following names are invalid: 4Foo, foo@bar, foo%bar.

2. Variable names must be unique throughout the XML document.

> *A validating XML editor will enforce the syntax.*

## 5.3 XML Expressions

The vocabulary and operators are used to create XML expressions. The framework allows for creating expressions of any complexity. These expressions are represented as valid XML and the validity is strongly checked against the Schema. An XML expression is represented in the form where the operator precedes the operands. This is called a Polish Notation—a typical computer representation of the expressions. Polish notation is used in a number of languages: Lisp, Forth, and Postscript.

### 5.3.1 Polish Notation

In the Polish Notation the operator precedes the operands. Here are some examples of how the conventional expressions are translated to Polish Notation:

| Conventional Expression | Polish Notation |
|---|---|
| **(1 + 2 + 3)** | (add 1 2 3) |
| **(5 * 6 * 7)** | (mul 5 6 7) |
| **((1 + 2 + 3) * (5 + 6 + 7))** | (mul (add 1 2 3) (add 5 6 7)) |
| **(a \| b)** | (or a b) |
| **(a \| b \| (~c))** | (or a b (not c)) |
| **(a & b & (~c))** | (and a b (not c)) |
| **(x == 5)** | (eq x 5) |
| **((x + y + z) == 5)** | (eq (add x y z) 5) |

### 5.3.2 XML Notation

XML notation used in ConflictDetector corresponds one-to-one to Polish Notation. Here are some the above examples represented in XML form:

| Conventional Expression | Polish Notation | Polish Notation |
|---|---|---|
| **(1 + 2 + 3)** | (add 1 2 3) | <int.add><br><int.const value="1"/><br><int.const value="2"/><br><int.const value="3"/><br></int.add> |
| **(5 * 6 * 7)** | (mul 5 6 7) | <int.mul><br><int.const value="5"/><br><int.const value="6"/><br><int.const value="7"/><br></int.mul> |
| **((1 + 2 + 3) * (5 + 6 + 7))** | (mul (add 1 2 3) (add 5 6 7)) | <int.mul><br><int.add><br><int.const value="1"/><br><int.const value="2"/><br><int.const value="3"/><br></int.add><br><int.add><br><int.const value="5"/><br><int.const value="6"/><br><int.const value="7"/><br></int.add><br></int.mul> |
| **(a \| b)** | (or a b) | <or><br><bool.ref idref="a"/><br><bool.ref idref="b"/><br></or> |
| **(a \| b \| (~c))** | (or a b (not c)) | <or><br><bool.ref idref="a"/><br><bool.ref idref="b"/><br><not><bool.ref idref="c"/></not><br></or> |
| **(x == 5)** | (eq x 5) | <int.eq><br><int.ref idref="x"/><br><int.const value="5"/><br></int.eq> |
| **((x + y + z) == 5)** | (eq (add x y z) 5) | <int.eq><br><int.add><br><int.ref idref="x"/><br><int.ref idref="y"/><br><int.ref idref="z"/><br></int.add><br><int.const value="5"/><br></int.eq> |

# 6 Lessons

In this section, we will demonstrate the ConflictDetector by walking through some examples. Though the examples are simple, most of the aspects of the technology are covered in this section.

The following examples assume that `UserGuide` folder is located in the `C:\XML` folder. Before unzipping UserGuide.zip, create a `C:\XML` folder, then unzip `UserGuide.zip` and place the contents in the `C:\XML` folder. The folder contains the following files:

- `ConflictDetector-UserGuide.docx`
- `ConflictDetectionConfig.xsd`
- `VersionOps.xsd`
- `XmlActions.xsd`
- `XmlExpressions.xsd`
- `XmlVocabulary.xsd`
- `*.xml - samples files`

## 6.1 Lesson 1: ConflictDetector Basics

This lesson uses `NotepadCalc.xml` found in the UserGuide folder.

### 6.1.1 Conflict Definition

In this simple use case, we allow Notepad to run only if Calculator is not running.

```
<Conflict id="NotepadCalcExclusion" Title="Allow Notepad only if Calc is not running">
  <IsApplicable>
    <IsFileExist FileName="C:\Windows\System32\notepad.exe"/>
  </IsApplicable>
  <ResolveConflict>
    <Triggers>
      <OnCondition>
        <IsProcessRunning ProcessName="calc.exe"/>
      </OnCondition>
    </Triggers>
    <Condition>
      <IsProcessRunning ProcessName="notepad.exe"/>
    </Condition>
    <Actions>
      <StopProcess ProcessName="notepad.exe"/>
    </Actions>
  </ResolveConflict>
  <RestoreState>
    <Triggers>
      <OnCondition>
        <not>
          <IsProcessRunning ProcessName="calc.exe"/>
        </not>
      </OnCondition>
    </Triggers>
    <Actions>
      <StartProcess ProcessName="notepad.exe"/>
    </Actions>
  </RestoreState>
</Conflict>
```

| XML Tag | Description |
|---|---|
| <IsApplicable> | The conflict is applicable only if file `C:\Windows\System32\notepad.exe` exists. Otherwise, this conflict is ignored. |
| <ResolveConflict> | Defines how to detect and resolve a conflict and contains the following elements:<br>▪ <Triggers> A general OnCondition trigger is used. It contains a Boolean expression that checks if calc is running.<br>▪ <Condition> The condition is evaluated to true if notepad is running. This indicates the conflict presence and the Actions are taken.<br>▪ <Actions> The action kills Notepad. |
| <RestoreState> | Used to define when and how to restore the original state (or reverse the previous actions) only if the conflict was actively resolved. It contains the following elements:<br>▪ <Triggers> When calc is not running the Condition is periodically checked.<br>▪ <OnCondition> If the condition defined in ResolveConflict is evaluated to false (i.e. if notepad is not running), the restore Actions are taken.<br>▪ <Actions> The restore action is to start Notepad. |

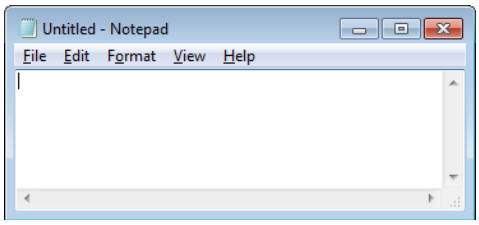
### 6.1.2 Testing the Sample

#### 6.1.2.1 Set ConflictDetectionConfig.xml

Copy the sample to your Open Mobile profile directory (the directory that contains `Engine.XML`) and rename it to `ConflictDetectionConfig.xml`. Start or Restart iPlatformService:

#### 6.1.2.2 Start Notepad

Make sure that Calc is not running. Start Notepad.

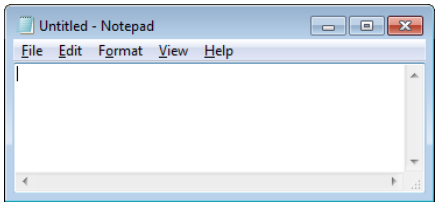



#### 6.1.2.3 Start Calculator

After Calc starts, within 2 sec Notepad is killed. The conflict is resolved and actively enforced.

#### 6.1.2.4 Close Calculator

Close calculator and observe that Notepad reappears.



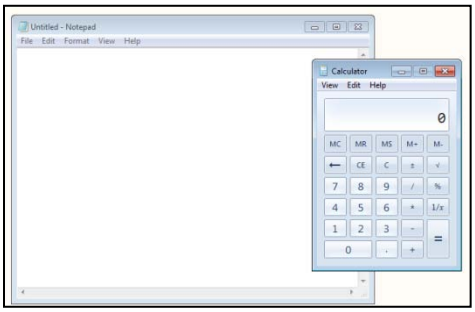

#### 6.1.2.5 Change ConflictDetector Poll Interval

Open `ConflictDetectionConfig.xml` and modify TimeInterval (located close to the top of the file) to 10 sec:

```
<Debug WatchConfigFile="true"/>
<PollInterval TimeInterval="00:00:10"/>
```

There is no need to restart the service because WatchConfigFile is set to true.

Start and close Calc a few times. Notice that Notepad is closed within 0 to 10 sec and reappears also within 0 to 10 sec.



#### 6.1.2.6 Restore TimeInterval.

Close Calc and make sure Notepad is running.

### 6.1.3 ConflictMonitor

ConflictMonitor is a tool for monitoring a conflict state.

#### 6.1.3.1 Enabling and Starting ConflictMonitor

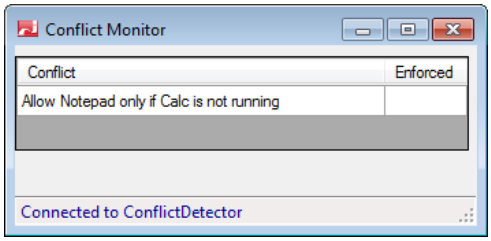
The default location for ConflictMonitor is:

```
C:\Program Files\iPass\Open Mobile\omsi\Plug-ins\iPass.ConflictDetector.Plugin\ConflictMonitor.exe
```

It is disabled by default. To enable ConflctDetector, create a dummy file `AllowConflictMonitor` in the profile directory. This file instructs ConflictDetector to establish a connection with ConflictMonitor.

Restart the service.

Start ConflictMonitor:



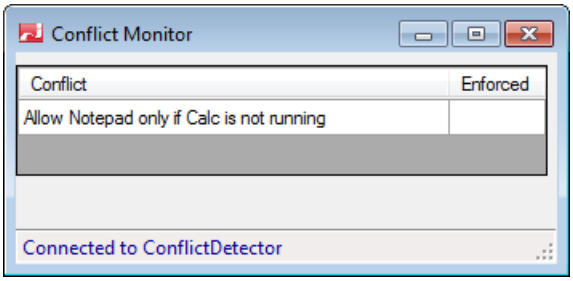

For the rest of the exercises keep ConflictMonitor open.

### 6.1.4 Experiments

In this long section we will perform multiple experiments that will refresh our basic ConflictDetector understanding.

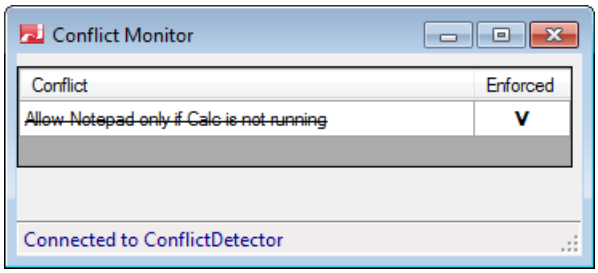
#### 6.1.4.1 Experiment #1: start Notepad

Start Notepad and notice that the conflict becomes present (it is not crossed-out).



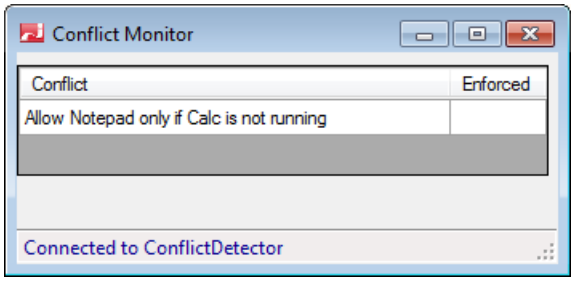

#### 6.1.4.2 Experiment #2: resolving the conflict

Start Calc and notice that ConflictDetector closes Notepad. The condition is actively resolved, as indicated by the checkmark **V** in ConflictMonitor under **Enforced**:



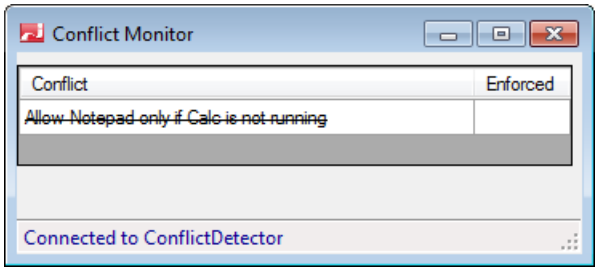

#### 6.1.4.3 Experiment #3: restoring the state

Close Calc and notice that ConflictDetector restores the state by re-launching Notepad.




#### 6.1.4.4 Experiment #4: making conflict inactive

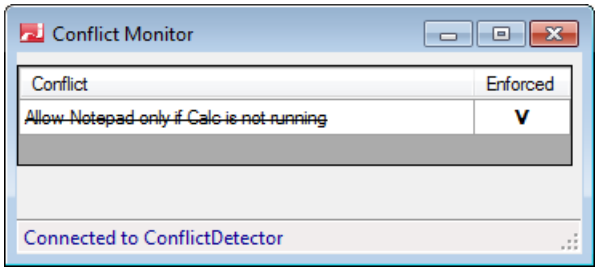
Close Notepad and notice that the conflict becomes inactive (crossed-out).



Start Calc and notice that nothing changes. Close Calc and notice that nothing changes. This shows that the state is restored only when the conflict is actively resolved.

#### 6.1.4.5 Experiment #5: continuous Notepad enforcement

Start Calc and then start Notepad again. Notice that ConflictDetector closes Notepad again.




#### 6.1.4.6 Experiment #6: restore the state

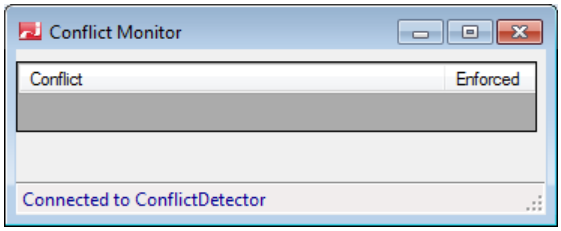
Restore the state by manually killing Calc.

#### 6.1.4.7 Experiment #7: IsApplicable=false

Modify the config file. Make IsApplicable false by setting a wrong Notepad path:

```
<IsApplicable>
  <IsFileExist FileName="C:\Windows\System32\WRONGPATH.exe"/>
</IsApplicable>
```

The config file is reloaded automatically after it is saved. Notice that the conflict disappears.



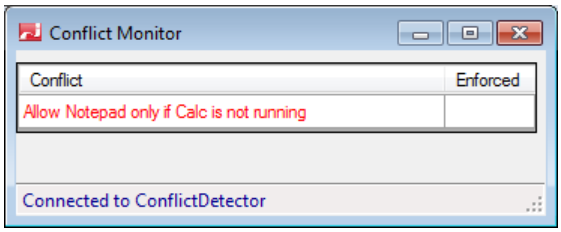

Restore the config file by providing a valid Calc path.

#### 6.1.4.8 Experiment #8: Failure to resolve a conflict

Remove ResolveConflict Actions:

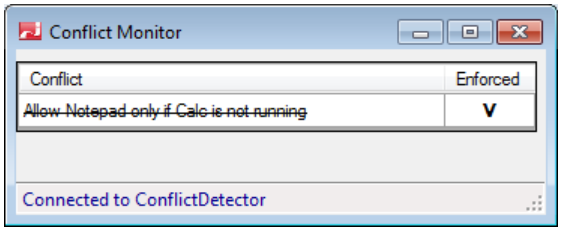
```
<ResolveConflict>
+   <Triggers>
+   <Condition>
    <Actions>
    </Actions>
</ResolveConflict>
```

Make sure that Notepad is running and then start Calc. Notice that the ConflictDetector was unable to resolve the conflict, as indicated in red.



#### 6.1.4.9 Experiment #9

Restore the config file and notice that the conflict becomes resolved.




#### 6.1.4.10 Experiment #10: Zombie state

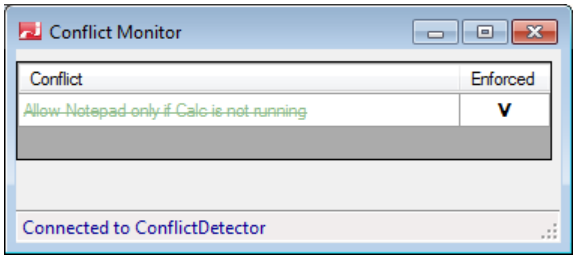
This is a more advanced topic that demonstrates what may happen after a profile update, when some conflict definitions are removed.

Make sure you are in the enforced state, as shown in the previous experiment.

Remove the Conflict from the config file:

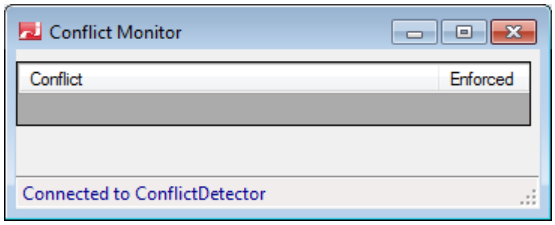
```
<Conflicts>
</Conflicts>
```

Notice the enforced conflict changes color:



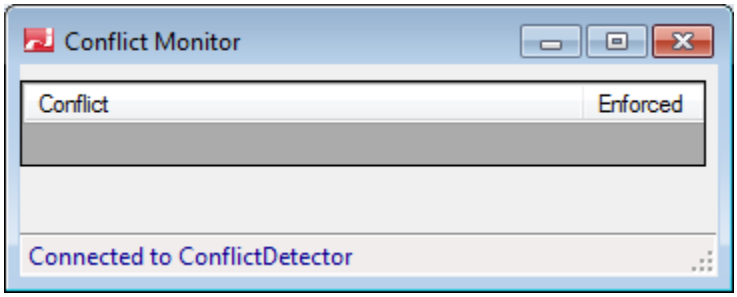

ConflictDetector just performed a non-trivial operation of matching the new config file with the current conflict state. Though the config file no longer contains the enforced conflict, ConflictMonitor does not remove the enforced conflict, but marks it for removal (or puts it in the **zombie** state).

Kill Calc and notice that the state is restored by launching Notepad. Also notice that the conflict is removed:




#### 6.1.4.11 Experiment #11: restore the state

Restore the config file and then enforce the state by starting Calc.



## 6.2 Lesson 2: Streamlining Scripts Using Resources

A complete example can be found in `NotepadCalc-Streamlined.xml`.

### 6.2.1 Divide and Concur

Divide-and-concur technique is the most proven programming technique for dealing with complexity. To simplify and streamline scripts, the administrator may define custom variables and custom actions in the Resources section of the script. This is very similar to defining procedures in a typical programming language. There are two main reasons to distill code into procedures:

1. When the block of code is repeated.
2. To simplify the logic of higher-level procedures by creating helper procedures (even if those procedures are used once only).

### 6.2.2 NotepadCalc Deficiency

NotepadCalc defined in the previous section has somewhat verbose trigger definitions:

```
<Triggers>
  <OnCondition>
    <IsProcessRunning ProcessName="calc.exe"/>
  </OnCondition>
</Triggers>

<Triggers>
  <OnCondition>
    <not>
      <IsProcessRunning ProcessName="calc.exe"/>
    </not>
  </OnCondition>
</Triggers>
```

### 6.2.3 Streamlining NotepadCalc

Following the best programming practices of reducing the complexity, we should reduce the trigger definitions to a single line. This is achieved by defining condition predicates as variables IsCalcRunning and IsCalcNotRunning resources. Using those definitions, the above triggers can be reduced to the following:

```
<Triggers>
  <OnConditionRef idref="IsCalcRunning"/>
</Triggers>

<Triggers>
  <OnConditionRef idref="IsCalcNotRunning"/>
</Triggers>
```

#### 6.2.3.1 IsCalcRunning Definition

IsCalcRunning is defined in Resources as:

```
<bool.def id="IsCalcRunning">
  <IsProcessRunning ProcessName="calc.exe"/>
</bool.def>
```

#### 6.2.3.2 IsCalcNotRunning Definition

IsCalcNotRunning can be defined in Resources as

```
<bool.def id="IsCalcRunning">
  <not>
    <IsProcessRunning ProcessName="calc.exe"/>
  </not>
</bool.def>
```

#### 6.2.3.3 <not.ref> Operator

Changing the logical polarity of a Boolean expression by using the <not> operator is a common practice. To streamline this operation, <not.ref> operator is defined. It is equivalent to <not> operator, except the reference to a previously defined predicate variable is used. Using this operator, IsCalcNotRunning can be defined as:

```
<bool.def id="IsCalcNotRunning">
  <not.ref idref="IsCalcRunning"/>
</bool.def>
```

### 6.2.4 Conflict Definition

Here is a complete snapshot of the streamlined definition. To reduce the size, some of the unchanged sections have been collapsed:

```
<ConflictDetectionConfig>
  <Resources>
    <Variables>
      <bool.def id="IsCalcRunning">
        <IsProcessRunning ProcessName="calc.exe"/>
      </bool.def>
      <bool.def id="IsCalcNotRunning">
        <not.ref idref="IsCalcRunning"/>
      </bool.def>
    </Variables>
+   <Actions>
  </Resources>
  <Conflicts>
    <Conflict id="NotepadCalcExclusion" Title="Allow Notepad only if Calc is not running">
+     <IsApplicable>
      <ResolveConflict>
        <Triggers>
          <OnConditionRef idref="IsCalcRunning"/>
        </Triggers>
+       <Condition>
+       <Actions>
      </ResolveConflict>
      <RestoreState>
        <Triggers>
          <OnConditionRef idref="IsCalcNotRunning"/>
        </Triggers>
+       <Actions>
      </RestoreState>
    </Conflict>
  </Conflicts>
</ConflictDetectionConfig>
```

### 6.2.5 Experiments

The reader is encouraged to independently run experiments similar to the previous lesson.

## 6.3 Lesson 3: User-Triggered Actions

A complete example can be found in `UserRequests.xml`.

The previous lessons resolved conflicts automatically—there was no user involvement. This is considered best practice, but there may be some cases where user involvement would be required.

### 6.3.1 Conflict Definition

- Resolve Conflict: Per user request: Start Calc and Disable WiFi
- Restore State: Per user request: Kill Calc and Enable WiFi
- In both cases inform the user of the actions taken.

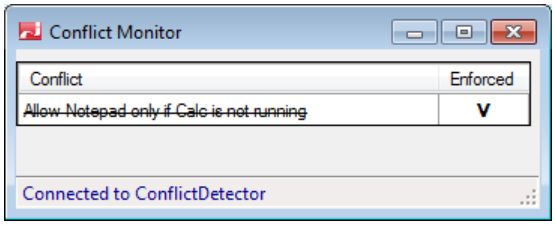
```
<Conflict id="UserTriggers" Title="User Actions">
  <IsApplicable>
    <bool.const value="true"/>
  </IsApplicable>
  <ResolveConflict>
    <Triggers>
      <OnUserRequest Message="Start Calc and Disable WiFi"/>
    </Triggers>
    <Actions>
      <StartProcess ProcessName="calc.exe"/>
      <UnmanageMedia MediaType="WiFi"/>
      <InformUser Message="Started Calc and Disabled WiFi"/>
    </Actions>
  </ResolveConflict>
  <RestoreState>
    <Triggers>
      <OnUserRequest Message="Kill Calc and EnableWiFi"/>
    </Triggers>
    <Actions>
      <StopProcess ProcessName="calc.exe"/>
      <ManageMedia MediaType="WiFi"/>
      <InformUser Message="Killed Calc and Enabled WiFi"/>
    </Actions>
  </RestoreState>
</Conflict>
```

Since the decision is not based on the state, the Condition is irrelevant and absent from the ResolveConflict element.

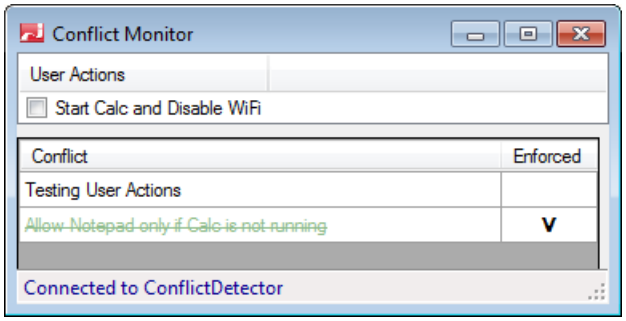
### 6.3.2 Experiments

#### 6.3.2.1 Experiment #1: Setting the Initial state

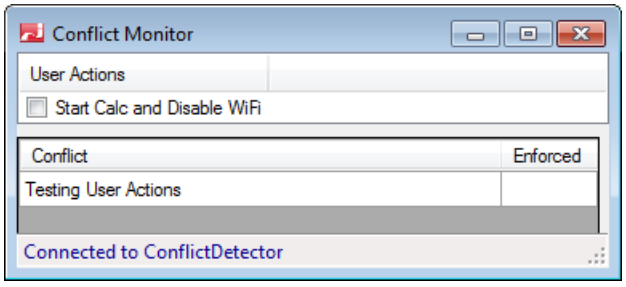
Make sure that ConflictDetector is in the state left after the previous lesson:




Copy `UserRequests.xml` to the profile folder and rename it `ConflictDetectionConfig.xml` **as an administrator**. Then, start `ConflictMonitor` (if not started yet).



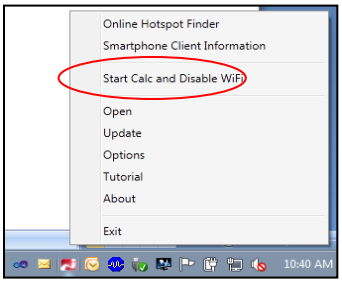

By performing the above actions we have simulated a profile update. As the result of this profile update a former conflict definition was removed, and since it was enforced, it has transitioned to Zombie state. Remove the Zombie state by either closing Calc or launching Notepad:




Notice “User Action” pane in ConflictMonitor that shows the text defined in ResolveConflict Trigger:

```
<OnUserRequest Message="Start Calc and Disable WiFi"/>
```

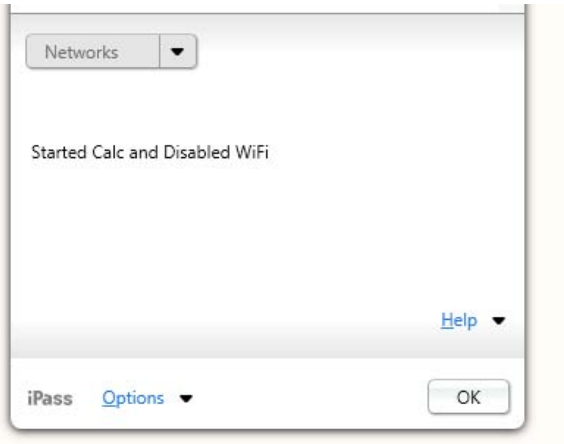
Right-click on the Open Mobile logo in the taskbar and notice that **Start Calc and Disable WiFi** was added to the menu (circled in red in the below screenshot).



#### 6.3.2.2 Experiment #2: Resolving the Conflict

Make sure Open Mobile UI window is visible. Resolve the conflict by selecting **Start Calc and Disable WiFi.**

Notice the user defined message in the Open Mobile user interface:




**Notice all 3 actions taken:**

1. Calc started.
2. Open Mobile disabled Wi-Fi.
3. The specified Inform Message is shown.

**Also notice the ConflictMonitor state:**

1. The conflict is resolved, as indicated by **V** mark.
2. The conflict User Request Message was changed to **Kill Calc and Enable WiFi**.

#### 6.3.2.3 Experiment #3: Restoring the State

In the taskbar right-click on the Open Mobile icon and select "**Kill Calc and Enable WiFi"**. Notice that the state is restored and the user-defined restore message is shown

## 6.4 Lesson 4: Learning the Simulation Environment

This lesson introduces the Simulation Environment. In this lesson we will also learn how to create more complex conditions. A complete example can be found in `DisableWiFi.xml`.

### 6.4.1 Conflict Definition

In this example Calc represents a 3rd party connection manager.

#### 6.4.1.1 Resolve Condition Definition

Condition:

- NetCalc1 network or NetCalc2 network is present.
- There is no network connection (on any of type of Media).

Actions:

- Disable Open Mobile to manage Wi-Fi.
- Launch Calc.

#### 6.4.1.2 Restore State Definition

When Calc is closed or on Wakeup

Actions:

- Enable Open Mobile to manage Wi-Fi

```
<Conflict id="DisableWiFiOnCalc" Title="Disallow OM to manage WiFi if Calc is  running">
  <IsApplicable>
    <IsFileExist FileName="C:\Windows\System32\calc.exe"/>
  </IsApplicable>
  <ResolveConflict>
    <Triggers>
      <OnNetworkPresent NetworkName="NetCalc1"/>
      <OnNetworkPresent NetworkName="NetCalc2"/>
    </Triggers>
    <Condition>
      <and>
        <IsConnectionStateEq ConnectionState="Disconnected"/>
        <IsMediaManaged MediaType="WiFi"/>
        <not> <IsProcessRunning ProcessName="calc.exe"/>  </not>
      </and>
    </Condition>
    <Actions>
      <UnmanageMedia MediaType="WiFi"/>
      <StartProcess ProcessName="calc.exe"/>
    </Actions>
  </ResolveConflict>
  <RestoreState>
    <Triggers>
      <OnWakeup/>
      <OnProcessNotRunning ProcessName="calc.exe"/>
    </Triggers>
    <Actions>
      <ManageMedia MediaType="WiFi"/>
      <StopProcess ProcessName="calc.exe"/>
    </Actions>
  </RestoreState>
</Conflict>
```

#### 6.4.1.3 IsApplicable

The conflict is applicable only if the 3rd party connection manager is installed, as checked by presence of file "C:\Windows\System32\calc.exe". Otherwise, the conflict is ignored.

#### 6.4.1.4 ResolveConflict

##### *6.4.1.4.1 Triggers*

```
<Triggers>
    <OnNetworkPresent NetworkName="NetCalc1"/>
    <OnNetworkPresent NetworkName="NetCalc2"/>
 </Triggers>
```

The triggers trigger if either NetCalc1 or NetCalc2 network is present.

We are using one of the forms of OnNetworkPresent. A complete definition is:

```
OnNetworkPresent(opt string NetworkName, opt bool Is8021x, opt bool IsNonBroadcast)
```

Where `opt` stands for optional. Here are some examples:

| | |
|---|---|
| OnNetworkPresent NetworkName="Owl" Is8021x="true" IsNonBroadcast="true" | Triggered when ssid=Owl is present and it is 8021x and it is a non-broadcast. |
| OnNetworkPresent IsNonBroadcast="true" | Triggered when any non-broadcast network is present |
| OnNetworkPresent | Triggered when any network is present |

##### *6.4.1.4.2 Condition*

```
<and>
   <IsConnectionStateEq ConnectionState="Disconnected"/>
   <IsMediaManaged MediaType="WiFi"/>
   <not>
      <IsProcessRunning ProcessName="calc.exe"/>
   </not>
</and>
```

This defines a composite condition. It is defined in the form suitable for XML, where the operator precedes the operands.

##### *6.4.1.4.3 Actions*

```
<Actions>
  <UnmanageMedia MediaType="WiFi"/>
  <StartProcess ProcessName="calc.exe"/>
</Actions>
```

The actions disable Open Mobile to manage Wi-Fi and start Calc.

#### 6.4.1.5 RestoreState

```
<RestoreState>
   <Triggers>
      <OnWakeup/>
      <OnProcessNotRunning ProcessName="calc.exe"/>
   </Triggers>
   <Actions>
      <ManageMedia MediaType="WiFi"/>
      <StopProcess ProcessName="calc.exe"/>
   </Actions>
</RestoreState>
```

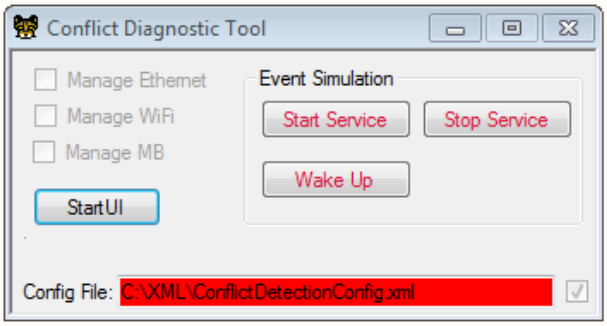
The state is restored on either wakeup or when Calc is closed.

### 6.4.2 Installing and Starting the Simulation Environment

To test `DisableWiFi.xml` sample, you need to be able to make the networks (NetCalc1 and NetCalc2) appear and disappear. This is where the simulation environment comes handy.

Before starting the simulation environment you must stop iPlatformService.

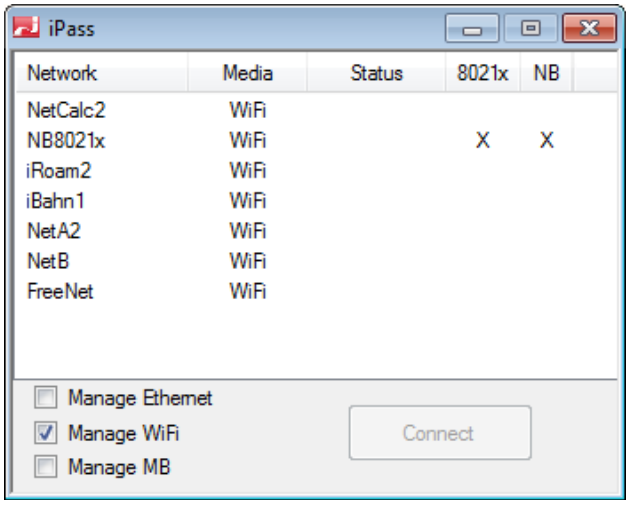
The simulation environment is installed automatically when starting `ConflictDiagnosticTool.exe`, which is located in `C:\Program Files\iPass\Open Mobile\omsi\Plug-ins\iPass.ConflictDetector.Plugin`. It must run as an Administrator. If this is your first time opening the tool, you will see a warning that there is no Config file and the following:




This shows that the default `C:\XML\ConflictDetectionConfig.xml` does not exist, and as a result, no conflicts are shown in ConflictMonitor.

After installing, verify that Diagnostic directory was created and that it contains the `Networks.txt` file.

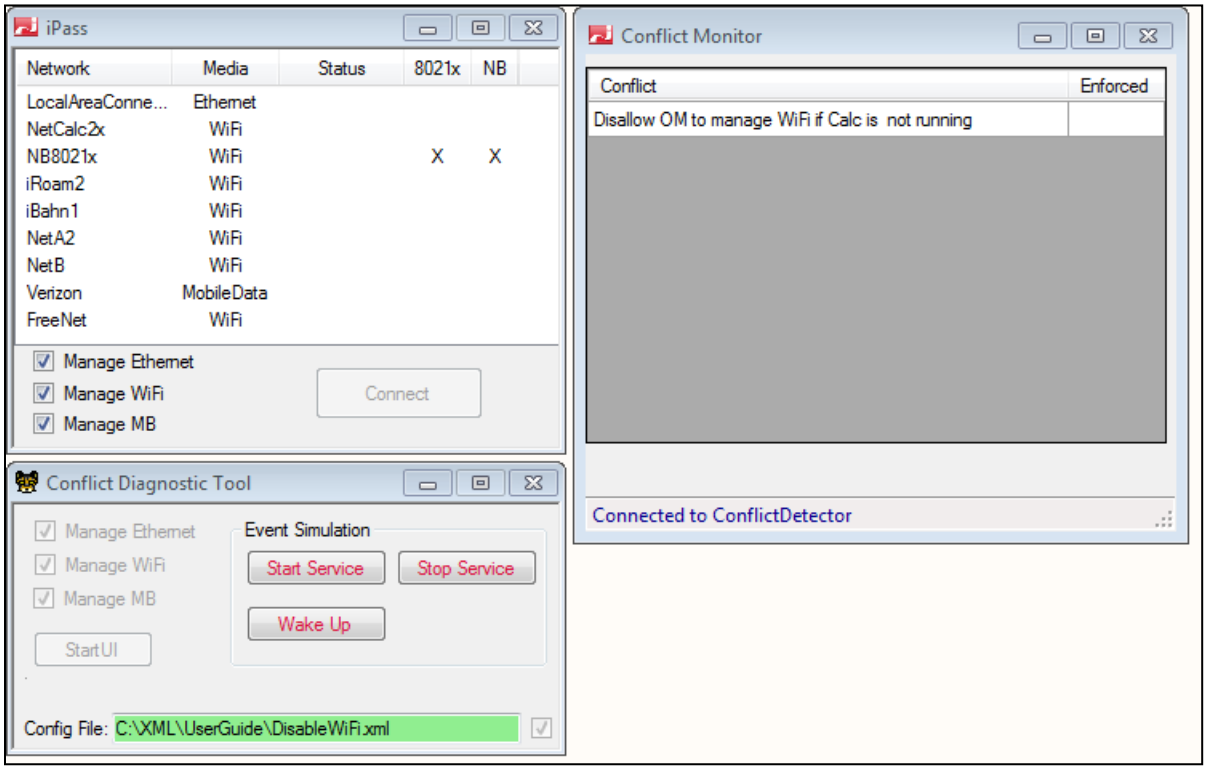
Check **Manage WiFi** and see the available networks (as defined in `Networks.txt`):



### 6.4.3 Experiments

#### 6.4.3.1 Experiment #1: Setting Initial state

1. Close `Calc` (if running).
2. Start `ConflictDetectionDemo` “as an administrator” (if not started yet).
3. Start `ConflictMonitor`.
4. Enter `DisableWiFi.xml` as a config file.
5. Click on Start UI button in `ConflictDetectionDemo`.




**Start UI** launces a simulation of some aspects of Open Mobile’s user interface (required for our simulation).

##### *6.4.3.1.1 Networks*

The network list comes from `Networks.txt` located in `C:\Temp\ConflictDetection`. Open the file in any of text editor and you should see the following:

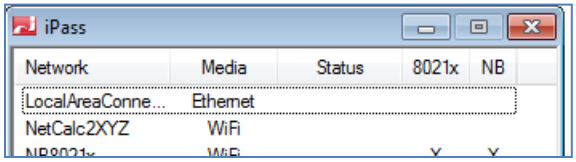
```
LocalAreaConnection:    Ethernet
NetCalc2X:              WiFi:
NB8021x:                WiFi:           8021x:          NB
iRoam2:                 WiFi:           owner=iPass:    Exclusive
iBahn1:                 WiFi:           owner=iPass
NetA2:                  WiFi:           owner=A:        Exclusive
NetB:                   WiFi:           owner=B
Verizon:                MobileData
FreeNet:                WiFi
```

Each line defines a network. Each line has the following format:

| NetworkName | Media | Optional Attributes |
|---|---|---|
| any SSID | Ethernet, WiFi, MobileData | owner, exclusive, 8021x, NB(non-broadcast) |

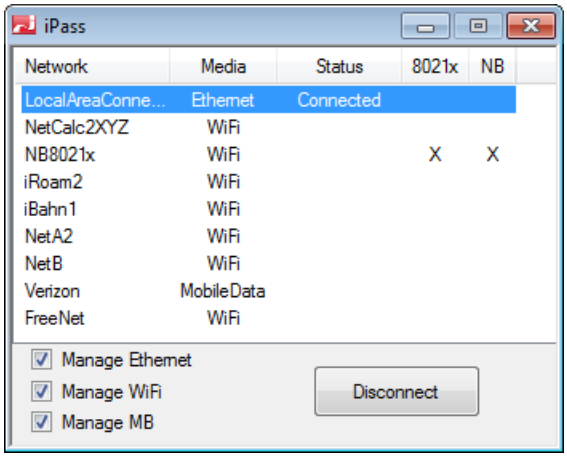
The entries are separated by column '**:**'. Owner and Exclusive are irrelevant for the current lesson and will be explained later.

Change NetCalc2X to NetCalc2XYZ and save the file. Note the immediate change in the network list:

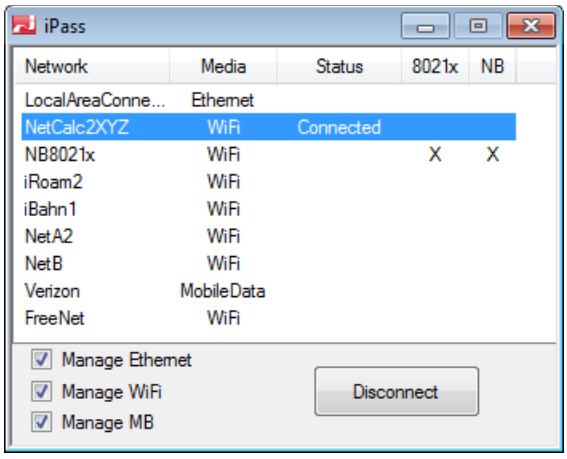



### 6.4.3.2 Experiment #2: Experimenting with UI

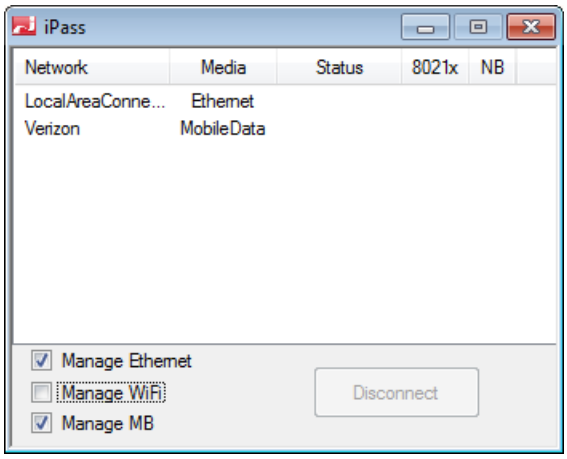
Connect to LocalAreaConnection by selecting the network and then selecting **Connect.** Note the result:



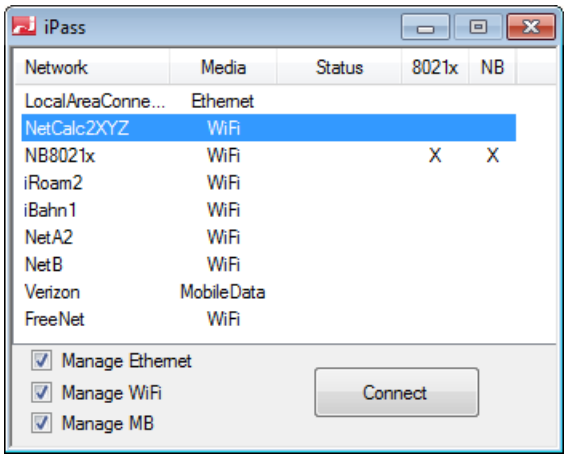

Connect to NetCalc2XYZ and note the result:



Uncheck **Manage WiFi**.



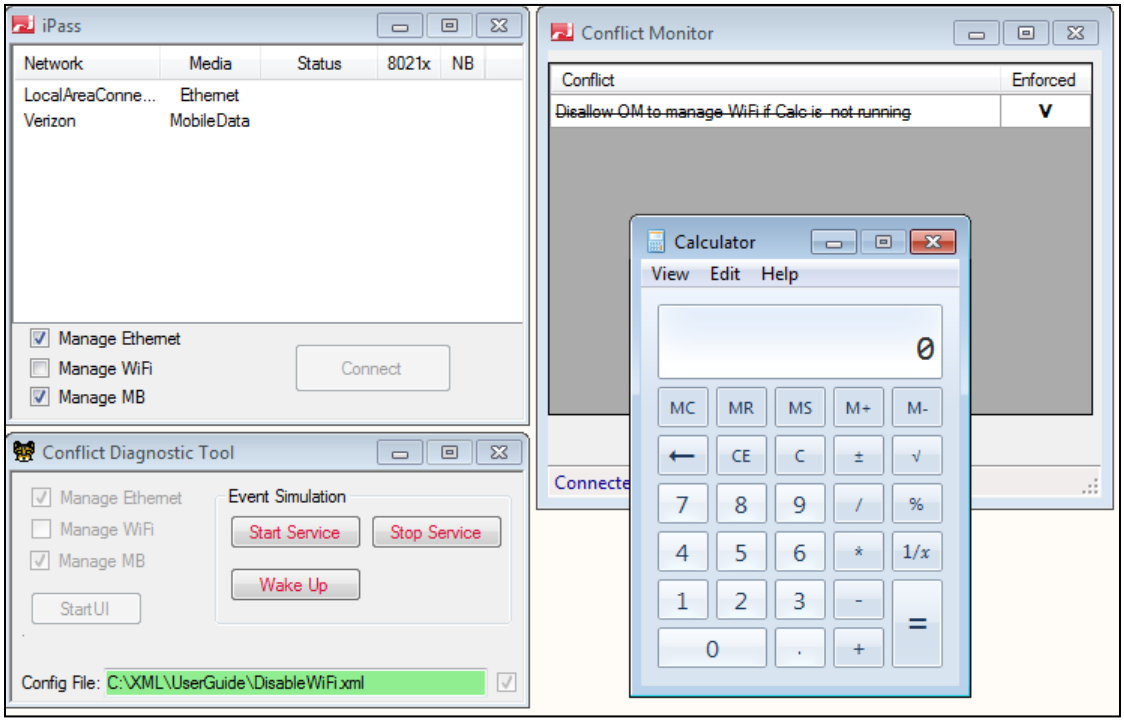

Check **Manage WiFi** and then disconnect from NetCalc2XYZ.



#### 6.4.3.3 Experiment#3: Creating and Resolving the Conflict

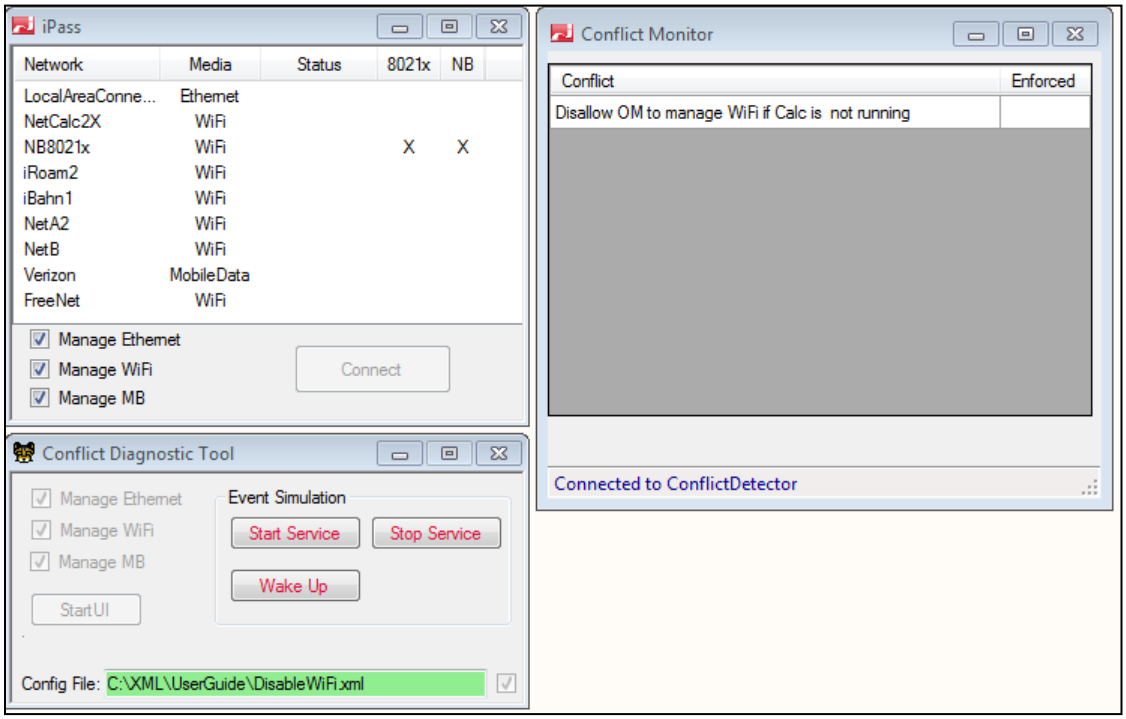
1. Make sure Open Mobile is managing WiFi.
2. Make sure Calc is not running.
3. In the text editor, change NetCalc2XYZ into NetCalc2.

Notice that this has triggered the Conflict, which was successfully resolved by disabling WiFi and launching Calc:



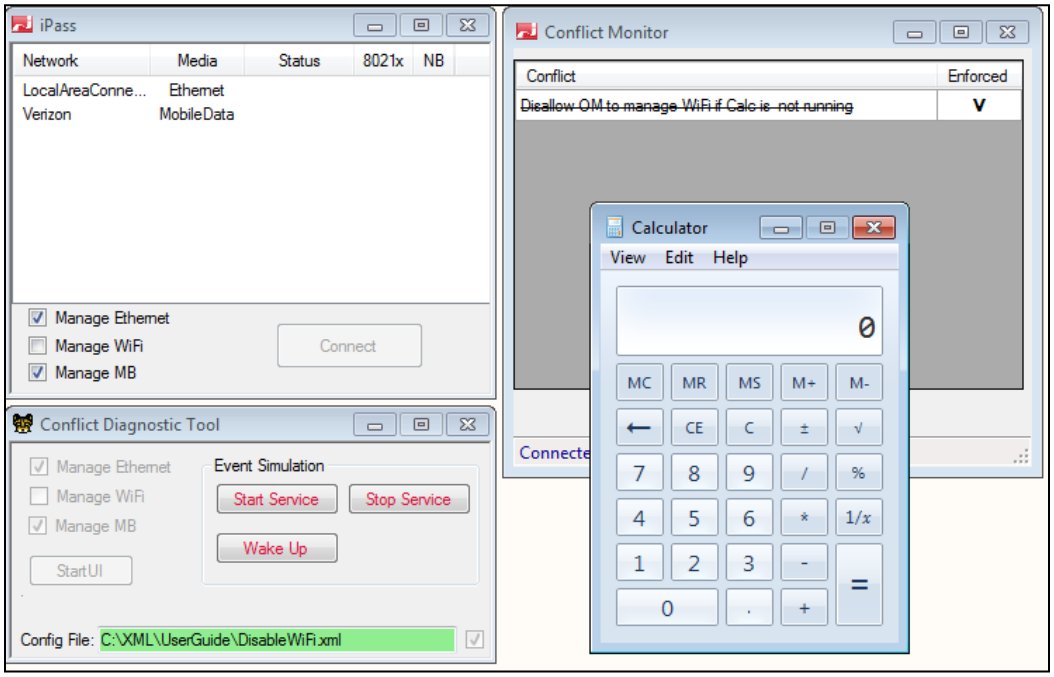
#### 6.4.3.4 Experiment#4: Restore the State

1. Rename NetCalc2 to NetCalc2X.
2. Kill Calc.
3. Notice that the state was restored:



#### 6.4.3.5 Experiment#5: Variation on Creating and Resolving the Conflict

1. Connect to any of the networks.
2. Rename NetCalc2X to NetCalc1.
3. Notice that the conflict was not triggered because we are still connected to a network.
4. Disconnect from the network.
5. Notice that the conflict was successfully resolved




#### 6.4.3.6 Experiment#6: Alternative Restore the State

Simulate Sleep/Wakeup sequence by pressing “Wake Up” button in ConflictDetectionDemo:

**Notice that:**

1. The state was restored for a fraction of a second (Calc was killed and networks reappeared).
2. This state resulted in the previous conflict.
3. The conflict was resolved (Calc reappeared and the networks disappeared)

#### 6.4.3.7 Experiment#7: Infinite Cycling Between Resolve and Restore

A logical error in conflict definition may result in a recursive cycling between resolve and restore. Let us demonstrate it.

1. Make sure that you are in the previously enforced state.
2. Connect to Verizon (this will prevent us from re-entering the conflict in the next step).
3. Kill Calc.
4. In the XML file, change RestoreState Triggers from OnProcessNotRunning to OnProcessRunning:

```
<RestoreState>
  <Triggers>
    <OnWakeup/>
    <OnProcessRunning ProcessName="calc.exe"/>
  </Triggers>
```

5. Disconnect from Verizon and watch the infinite Resolve/Restore sequence.
6. Stop the sequence by connecting to the Verizon network.
7. Restore the XML file to the original state.

## 6.5 Lesson 5: Self-Diagnostic and Script Debugging

A complete example can be found in `TestManageMedia.xml`.

### 6.5.1 Introduction

ConflictDetector provides a fairly rich vocabulary and powerful expressions. This functionality needs to be tested. The manual testing of each vocabulary item is cumbersome and unreliable. This is why some self-reflection mechanisms were added. The mechanisms include:

1. Special diagnostic operators: DebugMessage, DebugAssert, While
2. Extensive logging

In this lesson we will explain those mechanisms using `TestManageMedia.xml` sample.

### 6.5.2 The Sample

Since the sample is relatively large, let us split it to

- Custom Variables (Predicates)
- Custom Actions
- Resolve Conflict Section
- Restore State Section

#### 6.5.2.1 Custom Predicates

The Resource Variables section defines IsWiFiNetworksVisible and IsWiFiNetworksInvisible custom predicates:

```
<Variables>
  <bool.def id="IsWiFiNetworksVisible">
    <IsNetworkPresent MediaType="WiFi"/>
  </bool.def>

  <bool.def id="IsWiFiNetworksInvisible">
    <not.ref idref="IsWiFiNetworksVisible"/>
  </bool.def>
</Variables>
```

#### 6.5.2.2 Custom Actions

The Resource Actions section defines ManageWiFi and UnmanageWiFi actions. It refers to the previously defined predicates IsWiFiNetworksVisible and IsWiFiNetworksInvisible.

```
<Actions>
  <action.def id="ManageWiFi">
    <DebugMessage Message="#### ManageWiFi Test ###"/>
    <ManageMedia MediaType="WiFi"/>
    <DebugAssert Message="WiFi networks are unavailable when WiFi media is managed">
      <Condition>
        <bool.ref idref="IsWiFiNetworksVisible"/>
      </Condition>
    </DebugAssert>
    <DebugMessage Message="#### ---------- ###"/>
  </action.def>

  <action.def id="UnmanageWiFi">
    <DebugMessage Message="#### UnanageWiFi Test ###"/>
    <UnmanageMedia MediaType="WiFi"/>
    <DebugAssert Message="WiFi networks are visible when WiFi media is unmanaged">
      <Condition>
        <bool.ref idref="IsWiFiNetworksInvisible"/>
      </Condition>
    </DebugAssert>
  </action.def>
</Actions>
</Resources>
```

In the case of a ManageMedia test, DebugAssert statement asserts that the Wi-Fi networks are visible. This is true only in the environments where Wi-Fi networks exist (required to run this test).

In case of UnmanageMedia test, DebugAssert statement asserts that the Wi-Fi networks are invisible.

#### 6.5.2.3 Resolve Conflict

This is where the test is defined. The test starts when the user clicks **Start Test** in the Open Mobile UI or the ConflictMonitor. The test is surrounded by captions "######" and "=====".The test runs in While loop for 2 times and executes custom actions defined in the Resource section. After each action, it pauses for 1 sec. This allows the user to see the effect of these actions (such as W-Fi networks appearance anddisappearance).

```
<ResolveConflict>
  <Triggers>
    <OnUserRequest Message="Start Test"/>
  </Triggers>
  <Actions>
    <While MaxCount="2">
      <Actions>
        <DebugMessage Message="##################################################"/>
        <action idref="ManageWiFi"/>
        <Sleep Time="00:00:01"/>
        <action idref="UnmanageWiFi"/>
        <Sleep Time="00:00:01"/>
        <action idref="ManageWiFi"/>
        <Sleep Time="00:00:01"/>
        <DebugMessage Message="=========================================="/>
      </Actions>
    </While>
  </Actions>
</ResolveConflict>
```

#### 6.5.2.4 Restore State

In an unlikely event when the test fails (or the test is modified to change the state), this section restores the state by the user clicking **Restore State**.

```
<RestoreState>
  <Triggers>
    <OnUserRequest Message="Restore State"/>
  </Triggers>
  <Actions>
    <action idref="ManageWiFi"/>
  </Actions>
</RestoreState>
```

### 6.5.3 Experiments

The reader should run tests, note the results, and review the log to locate the messages defined by DebugMessage operators.

### 6.5.4 Debug Messages

At this time the debug messages are only outputted to the log. In the future, they may get outputted to other channels (such as the ConflictMonitor window).

### 6.5.5 Reading and Understanding the Logs

<Need to add this content>

# 7 Real Life Examples

## 7.1 Cisco NAM

A complete code can be found in `Nam.xml`.

### 7.1.1 Introduction

`Nam.xml` resolves the conflict between Cisco Network Access Manager (NAM) and Open Mobile. It disables NAM when the user is trying to connect using OM and re-enables NAM only when a network managed by NAM (IATNB in the sample) is visible.

NAM consists of NAM service (nam) and NAM UI (vpnui.exe).

`Nam.xml` demonstrates the benefit of separation between the main logic and resource definitions (predicates and actions).

### 7.1.2 Main Logic

```
<Conflict id="NAM" Title="NAM Conflicts with OM">
  <IsApplicable>
    <bool.ref idref="IsNamExist"/>
  </IsApplicable>
  <ResolveConflict>
    <Triggers>
      <OnUserRequest Message="Enable Off-Campus Mode"/>
      <OnInternetPreconnect/>
    </Triggers>
    <Condition>
      <bool.ref idref="IsNamServiceRunning"/>
    </Condition>
    <Actions>
      <action.ref idref="StopNam"/>
    </Actions>
  </ResolveConflict>
  <RestoreState>
    <Triggers>
      <OnUserRequest Message="Enable On-Campus Mode"/>
      <OnConditionRef idref="IsNamNetworkPresent"/>
      <OnServiceStop/>
    </Triggers>
    <Actions>
      <action.ref idref="StartNam"/>
    </Actions>
  </RestoreState>
</Conflict>
```

#### 7.1.2.1 IsApplicable

The conflict is applicable when NAM service exists.

#### 7.1.2.2 ResolveConflict

The conflict exists when NAM service is running. The state is enforced when

- The user enables off-campus mode using OM UI.
- When the user attempts to connect to any of the networks.

The state is enforced by executing StopNam procedure..

#### 7.1.2.3 RestoreState

The enforced state is restored when

- The user enables on-campus mode using OM UI.
- NAM network is present.
- When the service stops. This is especially important during OM update, where OM must restart.

The state is restored by executing StartNam procedure.

### 7.1.3 Custom Predicates

The custom predicates are self-explanatory.

```
<bool.def id="IsNamExist">
   <IsServiceExist ServiceName="nam"/>
</bool.def>
<bool.def id="IsNamServiceRunning">
   <IsServiceRunning ServiceName="nam"/>
</bool.def>
<bool.def id="IsVpnUIVisible">
   <IsProcessRunning ProcessName="vpnui.exe"/>
</bool.def>
<bool.def id="IsNamNetworkPresent">
   <IsNetworkPresent NetworkName="IATNB"/>
</bool.def>
```

### 7.1.4 Custom Actions

```
<action.def id="StartNam">
   <StartService ServiceName="nam"/>
</action.def>
<action.def id="StopNam">
   <StopService ServiceName="nam"/>
   <ReprobeNetwork MediaType="WiFi" NetworkName="IATNB"/>
</action.def>
```

StartNam just starts nam service.

StopNam is more involved. After stopping NAM service it executes ReprobeNetwork, which removes it from the network cache and re-probes the network.

## 7.2 OM / IPC Switching

A complete code can be found in `ipc.xml`.

### 7.2.1 Introduction

`ipc.xml` resolves the conflict between Open Mobile and IPC.

# 8 Advanced Topics

This chapter is intended to give a deeper understanding of ConflictDetector for those who create ConflictDetector scripts and those who debug the scripts.

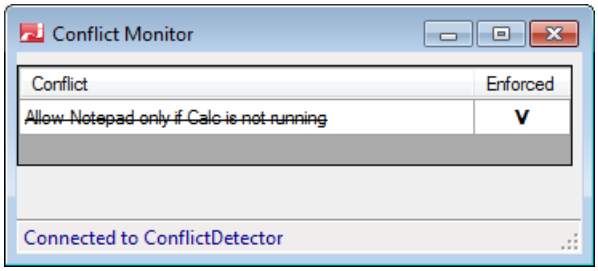
## 8.1 ConflictDetector as a State Machine

ConflictDetector defines a set of conflicts. Each Conflict definition is a state-machine.

**Here is a snapshot definition of a general state-machine:**

1. It defines an enumeration of states.
2. It always resides in a distinct state.
3. It is driven by events.
4. A state defines a set of transitions to other states, according to the events.
5. The state listens for a set of the events. If a proper event arrives, a predicate is executed. If it is evaluated to true, then the state transition occurs.
6. On entering an exiting a state, the state actions are performed.

ConflictDetector is a special state-machine. It has only two states: Enforced state and Unenforced state. As such, it can be called a Binary state-machine or a Boolean state-machine. This Binary/Boolean behavior is exploited by using conflict mnemonics to cross the conflict (~~conflict~~) when it is not present and use the check-mark to signify the enforcement:




**Translating the above 6 items for the ConflictDetector:**

1. The ConflictDetector defines two states: Enforced and Unenforced.
2. The ConflictDetector always resides in one of those two states.
3. Triggers serve as events
4. N/A
5. If a proper trigger is triggered, a Condition element is executed. If it is evaluated to true then the state transition occurs.
6. On **entering** a state, state Actions are performed.

## 8.2 Serial Processing Nature

ConflictDetector (as a typical state-machine) runs in a single thread. It listens to events only when it is in the idle state (not processing actions).

As a result of this serial processing, the events (triggers) can accumulate, which is undesirable. A well-behaved state-machine avoids the event accumulation by processing long-run actions in a separate thread while staying in a special state.

For example, ConnectionStateMachine while detecting AmIOn stays in CheckingAmIOn state. In this state it listens to all the events and acts properly. If an event is irrelevant, it is ignored. If it is relevant (such as a Disconnect Request), the state machine switches to the new state (e.g. Disconnecting). In either case the events are not accumulated.

Considering the Serial nature of ConflictDetector, it is recommended that users be cautious of long running operations. In that sense `TestManageMedia.xml`, though a good test example, is not a good example to follow when designing real Conflict scripts.