Avaya Aura® Contact Center Sample Application Guide:

# Soft Phone Attached Data

Issue 1.0 - August 2012

## About this sample application

This sample application is intended to demonstrate how to use the Avaya Aura® Contact Center (AACC) Communication Control Toolkit (CCT) .NET API to create a Web-based soft phone to manipulate call attached data.

In a contact center, it may be useful to associate a piece of data with a specific call. This would mean that if some information was gathered by an automated system or an agent, the data previously associated would be available with the call wherever the call is routed to afterward.

AACC is comprised of several different servers.

This sample application uses a .NET framework software development toolkit provided by Avaya which communicates with the Communication Control Toolkit server. CCT may or may not be co-resident with other AACC components on the same server.

This sample application will demonstrate basic generic CTI concepts as implemented by CCT, but is meant to highlight the attached data mechanisms available in the CCT .NET SDK.

## Table of contents

# Introduction

## Application features

The application is implemented as a Windows form hosted on a Web server. The interface can be used for basic call control (such as make call, answer call, transfer, etc), and also to set or retrieve call attached data.

A typical attached data scenario would be where an automated voice response system (IVR) answers a call, sets some attached data (usually retrieved via DTMF) and that attached data is then available to an agent when the call arrives in the contact center. The agent might then modify the attached data and transfer the call to another agent, call queue or third-party system.

Multiple instances of the application can be run on a single machine (multiple browser sessions).

## CCT concepts

Each CCT client starts an instance of the client toolkit. When credentials are provided to authenticate, the associated resource profile is allocated to the toolkit instance as a session.

A client is able to monitor (manipulate) any resources made available to it in its allocated session.

CCT implements a state model which terms any kind of call as a "Contact". So, in looking at specific parameters or function calls in the application source code for the purposes of this application, the word "contact"" is synonymous with a specific phone call.

In a contact center, in order to identify themselves, an agent uses their own unique ID to "login". This means the contact center can then manage that agents calls according to an internal stored profile associated with the unique ID provided. An agent can therefore be logged in or logged out.

# Application implementation

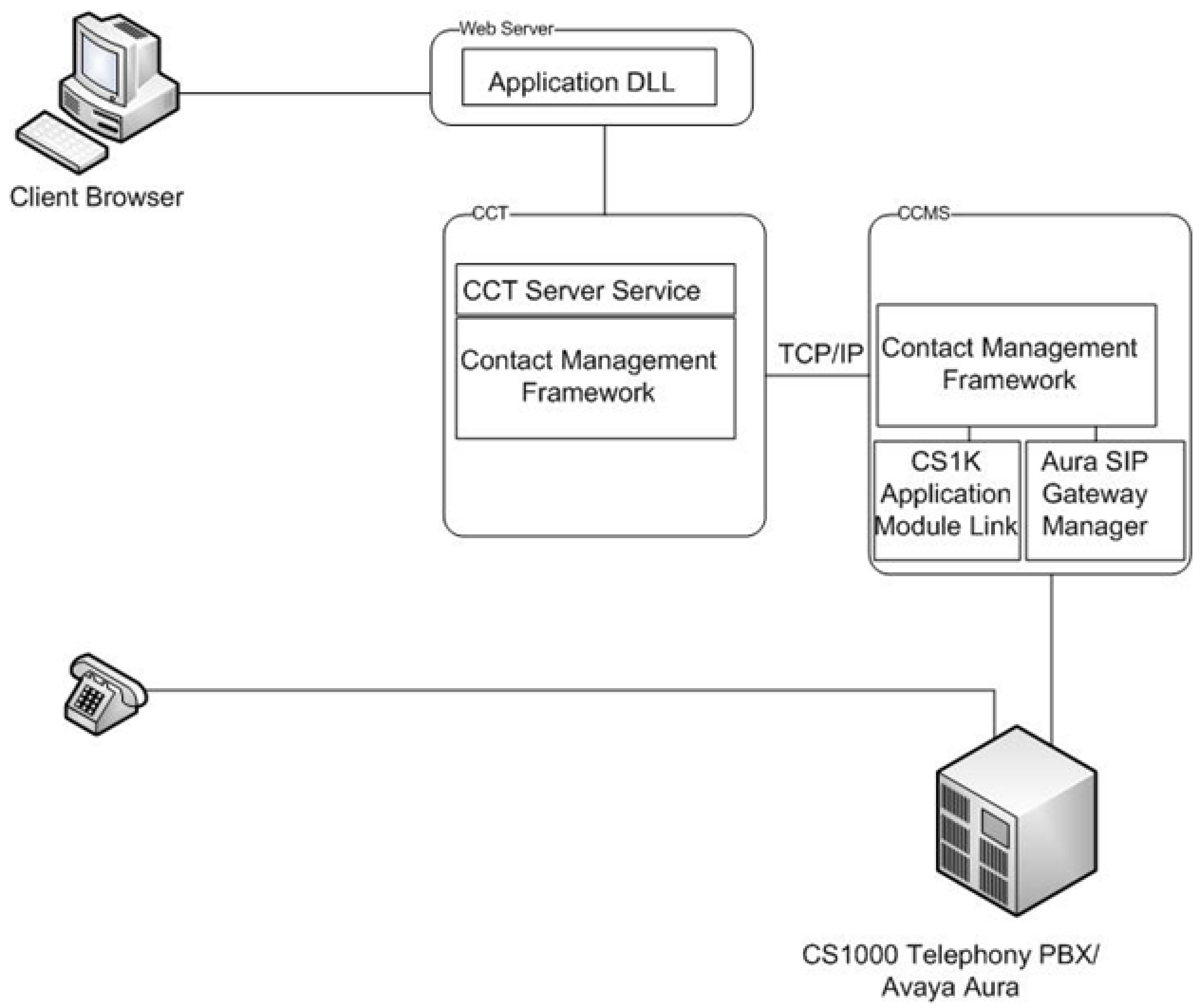
## Equipment and software used

- Telephony switch: Avaya Communication Server 1000 with Release 7.5 software
- Contact Center software: Avaya Aura Contact Center Release 6.2 Service Pack 5 RU03 (using associated client DLLs)
- Web server: Microsoft IIS 7
- Browser: Firefox 13.0.1, Internet Explorer 8.0
- Application Client computer: Windows 7 with Service Pack 1
- The application was written in C# (Visual Studio 2008) running .NET 3.0.

Note the application will run against any of the following telephone switch types:

- Avaya Communication Server 1000 Release 7.5
- Avaya Communication Server 1000 Release 7.5 running SIP Line Gateway
- Avaya Aura Release 6.1 (comprising the Avaya Aura Session Manager, Avaya Aura Communication Manager, Avaya Aura Enablement Server)

The telephony switch implementation is masked by Avaya Aura Contact Center software.

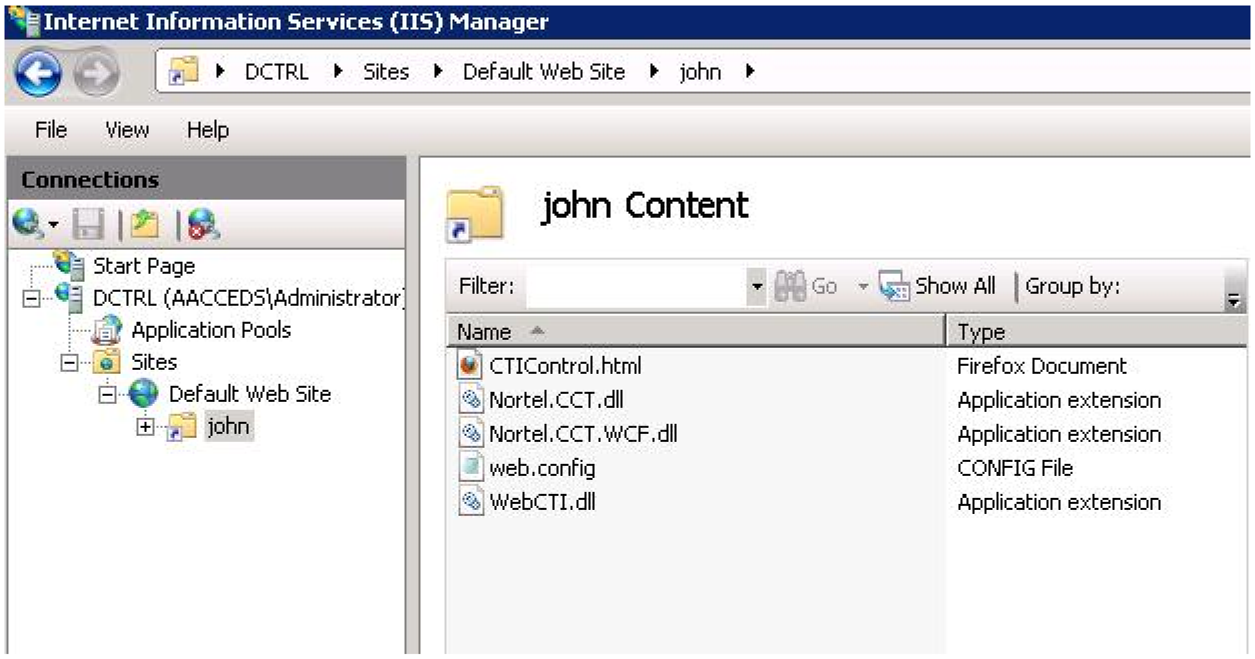

Client Browser
Web Server
Application DLL
CCT
CCT Server Service
Contact Management Framework
TCP/IP
CCMS
Contact Management Framework
CS1K Application Module Link
Aura SIP Gateway Manager
CS1000 Telephony PBX/ Avaya Aura

## Architecture

The application is implemented as a DLL hosted on a Web server.

On the Web server there is a small HTML file which invokes the DLL:

```
<html>
   <body>
      <p>Test Control</p>
   </body>
   <object id="WebCTI1" classid = "http:WebCTI.dll#WebCTI.WebCTICtrl"
            height="700" width="500" VIEWASTEXT></object>
</html>
```



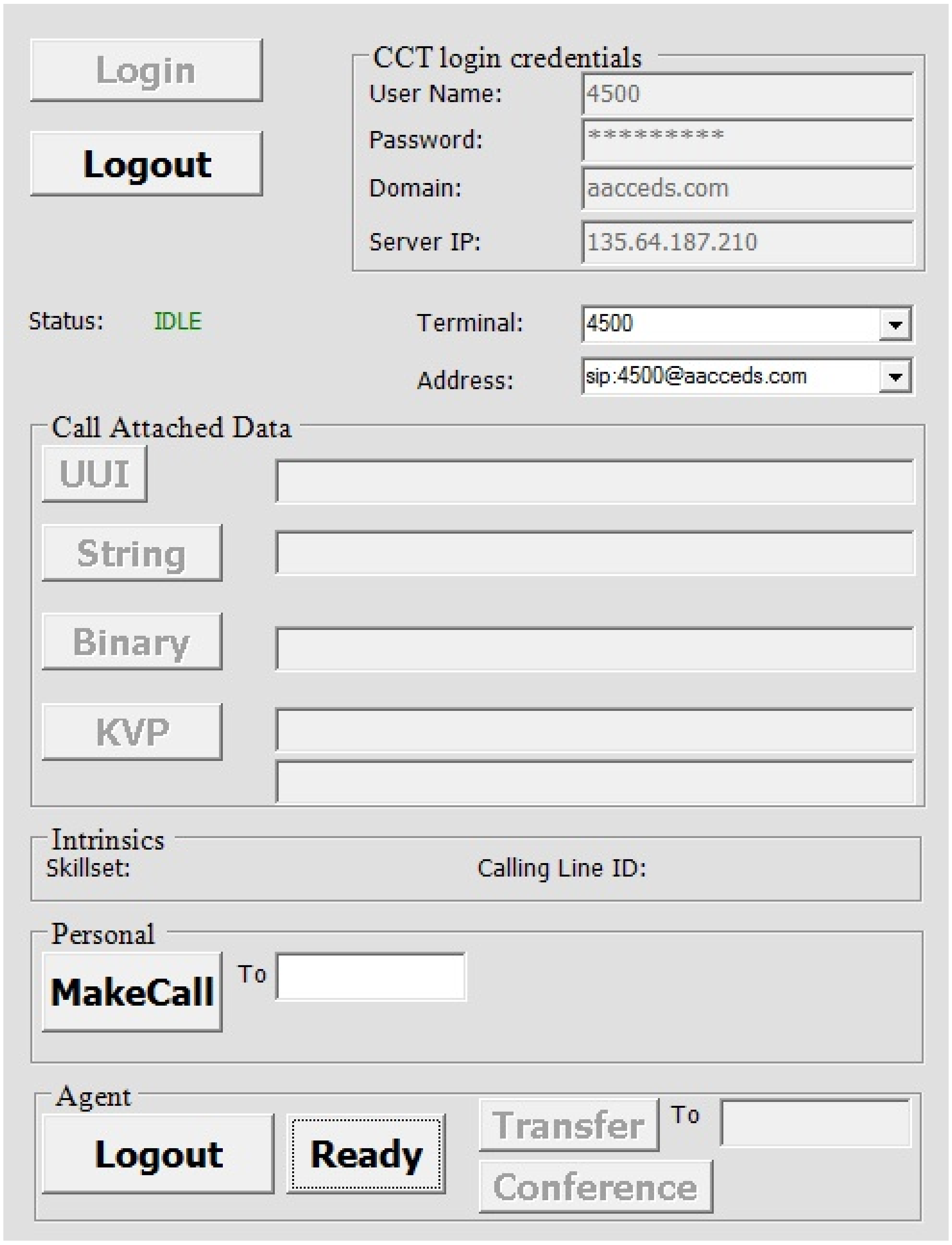
| Name | Type |
|---|---|
| CTIControl.html | Firefox Document |
| Nortel.CCT.dll | Application extension |
| Nortel.CCT.WCF.dll | Application extension |
| web.config | CONFIG File |
| WebCTI.dll | Application extension |

The user invokes a browser and enters the URL to the HTML file shown above. When the user clicks `Login`, the application initializes itself by running the `mLogin_Click` function.

This function does several things:

1. The credentials are taken from the GUI form and used to set up a session with the CCT server. This session contains a profile of CCT resources (addresses, agents) associated with the user credentials provided.
2. The application registers 3 event handlers to be notified of any changes to the state of any of the resources available in the session resources allocated. The application also registers a fourth event handler to be notified of a server disconnect (due to an inactivity timeout or network issues).
3. The application makes a list of non-multimedia lines and then assigns one of these lines to any agent in the session profile who does not have a default (static voice terminal) address already assigned.
4. The first available line is set as the chosen line, and it is then checked to see if an agent is logged in on that line, or if there is already a call on that line.
5. The GUI is updated accordingly.

After a successful login, the following form will be displayed in a Web browser:



The application caters for attached data is several formats.

A call can have 2 items of attached data associated with it (an instance of UUI data and an instance of regular attached data):

- UUI data is call related data which arrives with the call embedded in the call signaling. UUI data is not available to an application in a non-SIP environment.
- Regular attached data is available in one of 3 formats: string, binary and key value pair.

In the form above, if there is attached data associated with a call it will be displayed next to the appropriate type.

A user can edit this attached data by clicking on the button showing the relevant type and then setting the data.

A user is able to change either the address being monitored by the application or the line (terminal) being monitored by the application. When the line is changed the chosen address will automatically update to the first available address associated with that line.

When as address is changed, the `CheckAddressState` function is called to check the existing state of the address and update the GUI accordingly.

As events are received from the CCT server, the GUI is updated by the 4 event handlers the application registered for. The functions triggered on receiving an event from the CCT server are `OnAgentStateChange`, `OnConnectionStateChange`, `OnContactStateChange` and `OnDisconnect`.

The application uses the following triggers in the event handlers to be notified of a change:

**`OnAgentStateChange`**: `AgentProperty` enumerated types `ReadyStatu` and `LoggedIn` are specified in the event received.

**`OnConnectionStateChange`**: `TerminalConenctionState` enumerated types `Active`, `Bridged`, `InUse`, `Held`, `Idle`, `Unknown` and `Dropped` are specified in the event received.

**`OnContactStateChange`**: `ContactProperty` enumerated types `AttachedData` and `UserUserInfo` are specified in the event received.

**`OnDisconnect`** is triggered when the connection to the server is lost.

The application also implements handlers for the available buttons (note the button labels change dynamically on state change to update functionality available to the user – for example the button labeled `Login` will change to show `Logout` on receiving successful login notification from the server through the event handler).

As part of the various state updates (and also using the buttons on the GUI) the attached data is manipulated on active calls.

In order to create (or set) attached data, a new attached data object is created and then instantiated with the value of the data from the GUI. The value from the GUI first has to be translated to the appropriate data type. For example:

```
AttachedData tempBData = new
      AttachedData(System.Text.Encoding.UTF8.GetBytes(mBinaryt.Text));
try
{
   mConnect.Contact.Data = tempBData;
```

The contact data is then set to this object value.

In order to update the attached data displayed in the GUI at various points in the code (such as the event handlers) the application calls a function `CheckData`.

This application takes an attached data object, and then translates the attached data to a readable form. Next the application updates the GUI appropriately with the translated form. For example to convert the binary data the following code is used:

```
StringBuilder text = new StringBuilder(1024);
foreach (byte b in mData.BinaryData)
      text.AppendFormat("{0} ", b.ToString("X2"));
```

The application caters for both regular attached data (Binary, String or KVP) and also UUI data.

The 3 types of regular data are different forms of the same data item, so a call can only ever have 1 of the 3 types of regular attached data associated with it.

## Design Choices

Within the CCT .NET SDK for certain key concepts, there would be more than one functional mechanism available to a client application to implement.

This would mean that according to load, functional requirement or configuration required, an application developer could then choose the most suitable mechanism to implement the functions required.

## Event Handlers

The toolkit provides 16 delegates to allow an application to register for unsolicited events from the CCT server using event handlers.

This sample application implements 3 event handlers against its session object – Terminal Connection State, Contact Property and User Property. These trigger events in the application on changes in Terminal Connection state, Contact state and Agent state.

`Note`: It is recommended to use the `CreateGUIEventHandler` when registering for events. This prevents lock ups in a client application GUI caused by the timing of event state changes.

## Default agent login address

In order for an application to login an agent, the API requires that every agent has a ‘default’ address associated with it. This is referred to as the “`StaticVoiceTerminal`” for each agent. When an agent logs in, the agent will be logged in automatically on the default address. In this application, the default address is set for each agent on startup and when the agent logs out.

`Note`: This default address is stored on the server, and it is possible for any other application to set it.

## Agent URI / Position Id

While the application will work against AML CS 1000 PBX, SIP CS 1000 PBX or a SIP Avaya Aura PBX, the way individual line addresses are referenced is slightly different.

In a CS 1000 environment, for a specific line there is normally an agent key (to receive agent calls – no outbound functionality) and a DN key (for regular calls).

In a SIP environment there is only 1 address per line, this address is a SIP URI address.

The CCT .NET SDK uses type polymorphism to model the address, so to check if an address is agent capable the sample application checks the address type.

The SIP URI property is not set in a non-SIP CS 1000 environment, so the application can detect the environment that way.

## Attached data checking

In the application, a check is only made for attached data when a call is connected (answered, for example). There is no check for the presence of attached data on the call ringing (alerting) event. This is because it may take a very small amount of time for networked calls to retrieve the actual attached data from a remote source and associate the data with the call in the server. It is recommended to check for the presence of attached data on call answered events.

However, the `OnContactStateChange` event handler may retrieve the attached data before the call is answered as a result of a change of attached data event being triggered when the call arrives.

## Outbound calls

When a call ringing event is received the application determines if the call has been initiated from the selected address or if it is being answered on the selected address. This means either the selected address is receiving ringback from a far end phone or the address itself is ringing. This has an impact on how the application can manage the call subsequently.

The application checks if the call is inbound or outbound by checking the associated contact calling address as follows:

```
if(mState==TerminalConnectionState.Ringing)
{
    // Abandon outbound or Answer inbound
    if (mConnect.Contact.CallingAddress == chosenAddress.Name) // Outbound
   {
```

There is also a different way of handling outbound calls in a SIP environment.

In a non-SIP CS 1000 environment each line can have 2 addresses. There is a dedicated address for agent functionality and another address for non-agent calls. In this environment it is not possible to initiate an outbound call from an agent address.

In a SIP environment, the same address is used to perform agent functionality and originate and receive non-agent calls.

In this application in a SIP environment all inbound calls ring on the agent button, while any outbound calls are initiated from the `Make Call` button.

In a non-SIP environment when an agent address is selected the `Make Call` button is disabled. When a non-agent address is selected the `Agent` button is disabled.

`Note`: In a SIP environment it is possible for a non-agent call to be received even when the agent is set to not ready.

### Transfer and conference

The way in which transfer and conference are implemented differs slightly between SIP and non-SIP systems.

Blind transfer is not supported in SIP systems.

In this application, the code does not aim to manage supervised transfers or conferences. Rather, when a conference or transfer is initiated, the application will automatically try and complete the transfer or conference subsequently.

The application waits for a change of state reason on the monitored address which shows the conference or transfer has been successfully initiated. This event is triggered in the `OnConnectionStateChange` function.

In SIP this state change reason is `ConsultInitiated` for both conference and transfer.

In non-SIP environments the state change reason is `TransferInitiated` or `ConferenceInitiated`:

```
if (((bSIPPBX) && (e.Reason == Reason.ConsultInitiated)) ||
       ((!bSIPPBX) && (e.Reason == Reason.ConferenceInitiated)))
…
if (((bSIPPBX) && (e.Reason == Reason.ConsultInitiated)) ||
       ((!bSIPPBX) && (e.Reason == Reason.TransferInitiated)))
```

In non-SIP implementations a conference cannot be completed unless the conference party has answered.

For this application in non-SIP CS 1000 environments, it is recommended to configure auto answer on the conference target party using the AAA (automatic answer allowed) and HFA (hands free allowed) class of service on the set. This means as soon as the conference is initiated to the set it will be automatically answered by the PBX. The code will then complete the conference. If this feature (auto answer) is not configured, make sure the AHA (automatic hold allowed) feature is enabled on the CS 1000, this will allow the application to automatically drop the new conference leg of the call to retain control of the original call if the conference completion fails. The application does this by invoking “unhold” on the original call. The PBX then automatically drops the conference leg of the call.

### Incoming attached data

It is possible to provide PBX Caller information (such as UCID or CLID) as attached data on incoming calls.

In a non-SIP CS 1000 environment this is done through AACC scripting.

In a SIP Aura environment this is done through Communication Manager configuration.

## Application scope

The CCT .NET SDK provides a large and comprehensive suite of functionality. This application does not implement all functions available in the SDK, but rather those it requires to facilitate basic CTI and call attached data.

### Voice only media control

AACC does not only manage voice calls. It also has the facility to route and manage other media types such as email or IM.

While the Toolkit does provide certain functionality for all media types (such as alerting on a new contact) this application only processes voice calls.

### Agent functionality

This application only implements very basic agent functionality. It does not handle Not Ready reason codes, supervisor or emergency call, etc.

### Change of Static Voice Terminal

There is a `VoiceTerminal` agent property member which could be used to manage a scenario whereby another application has changed the default address (Static Voice Terminal) for an agent. This functionality is not covered in this application.

It is recommended that no other CTI application be run concurrently if that CTI application has access to the same addresses being controlled by this sample application.

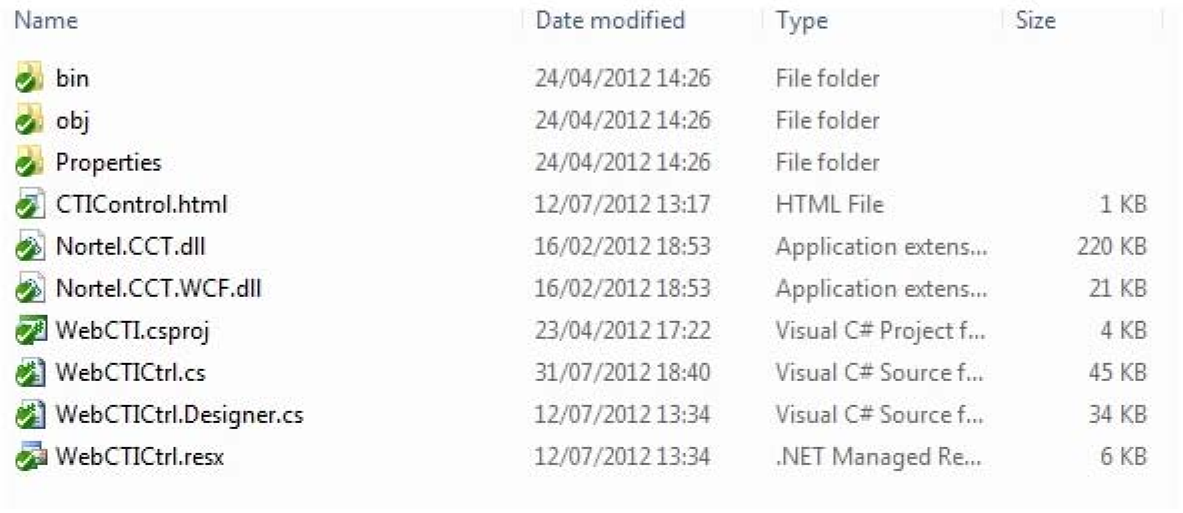
It is possible to run other CTI applications concurrently to monitor other addresses on the same PBX.

### Key value Pair attached data

This application only retrieves or sets the first pair of values in a Key Value Pair attached data object. There can be multiple pairs in the attached data object.

## Sample application file listing

The files included in the sample application are shown below:



| Name | Date modified | Type | Size |
|---|---|---|---|
| bin | 24/04/2012 14:26 | File folder | |
| obj | 24/04/2012 14:26 | File folder | |
| Properties | 24/04/2012 14:26 | File folder | |
| CTIControl.html | 12/07/2012 13:17 | HTML File | 1 KB |
| Nortel.CCT.dll | 16/02/2012 18:53 | Application extens... | 220 KB |
| Nortel.CCT.WCF.dll | 16/02/2012 18:53 | Application extens... | 21 KB |
| WebCTI.csproj | 23/04/2012 17:22 | Visual C# Project f... | 4 KB |
| WebCTICtrl.cs | 31/07/2012 18:40 | Visual C# Source f... | 45 KB |
| WebCTICtrl.Designer.cs | 12/07/2012 13:34 | Visual C# Source f... | 34 KB |
| WebCTICtrl.resx | 12/07/2012 13:34 | .NET Managed Re... | 6 KB |

Note: Replace the CCT DLLs with the appropriate version depending on the AACC version the application will run against. These DLLs are available on the AACC server.

## Appendix

### Glossary

AACC – Avaya Aura Contact Center
CLID – Calling Line ID
CTI – Computer Telephony Integration
DLL – Dynamic Link Library
DTMF – Dual-Tone Multi-Frequency signaling (telephone keypad tones)
IVR – Integrated Voice Response System
PSTN – Public Switched Telephone Network
UCID – Universal Call ID
UUI – User to User Information

### References

Avaya Aura® Contact Center CCT SDK Reference Release 6.2 Service Pack 5

---

Please e-mail any questions or comments pertaining to this sample application guide, along with the full title, directly to the Avaya DevConnect Program at devconnect@avaya.com.