Avaya Aura Contact Center

CODE

Table of Contents

Chapter 5 - Use Case – Monitor Real Time Statistical Data with RTD API

```
#include <nirtdtyp.h>
#include <nirtdapi.h>

ULONG rc = NIrtd_eOK;

// authorization structure used by login and
// name cache
NIrtd_tAPIauth authInfo = Nirtd_NullAuth;;

// Perform login to the server
  rc = NIrtd_login(&authInfo, ipAddress,
    username, password);

if (rc != NIrtd_eOK)
  {
    _tprintf(_T("\nNIrtd_login - failed to login: return code = %d"),
rc);
    return;
  }
  else
  {
    _tprintf(_T("\nNIrtd_login - success : return code = %d"), rc);
  };

// setup name cache for SkillsetID to SkillsetName translation
  rc = NIrtd_getNameCacheforDataColumn(&authInfo,
NIrtd_SKLST_SKILLSET_ID);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_getNameCacheforDataColumn failed : rc = %d"),
rc);
  rc = NIrtd_logout(&authInfo);
  return;
  }
  else
  {
  _tprintf(_T("\nNIrtd_getNameCacheforDataColumn succeeded."));
  }


  // allocate and initialize the query structure
  rc = NIrtd_allocateQuery(&query, NIrtd_INTRVL_SKLST);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_allocateQuery failed : rc = %d"), rc);
  rc = NIrtd_logout(&authInfo);
  return;
```

```
  }
  else
  {
  _tprintf(_T("\nNIrtd_allocateQuery succeeded."));
  };

  // select skillset ID as the first column
  rc = NIrtd_selectColumn(&query, NIrtd_SKLST_SKILLSET_ID);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_selectColumn failed : rc = %d"), rc);
  rc = NIrtd_freeQuery(&query);
  rc = NIrtd_logout(&authInfo);
  return;
  }
  else
  {
  _tprintf(_T("\nNIrtd_SKLST_SKILLSET_ID Column selected."));
  }

  // select number of agent available as the second column
  rc = NIrtd_selectColumn(&query, NIrtd_SKLST_AGENT_AVAIL);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_selectColumn failed : rc = %d"), rc);
  rc = NIrtd_freeQuery(&query);
  rc = NIrtd_logout(&authInfo);
  return;
  }
  else
  {
  _tprintf(_T("\nNIrtd_SKLST_AGENT_AVAIL Column selected."));
  }


  // start data stream
  rc = NIrtd_startDataStream(&authInfo, &query, refresh_rate * 1000,
callback_function, (void *)NULL, &requestId);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_startDataStream failed : rc = %d"), rc);
  rc = NIrtd_freeQuery(&query);
  rc = NIrtd_logout(&authInfo);
  return;
  }
  else
  {
  _tprintf(_T("\nNIrtd_startDataStream succeeded."));
  }
```

```
// one-time data propagation
  rc = NIrtd_singleDataRequest(&tableGrpPtr, &authInfo, &query);
  if (rc != NIrtd_eOK)
  {
  _tprintf(_T("\nNIrtd_singleDataRequest failed : rc = %d"), rc);
  rc = NIrtd_freeQuery(&query);
  rc = NIrtd_logout(&authInfo);
  return;
  }
  else
  {
  _tprintf(_T("\nNIrtd_singleDataRequest succeeded."));
  }

// call back function
ULONG callback_function(ULONG return_code, NIrtd_tRequestId requestId,
NIrtd_stTableGroup *tableGroup, void *yourPointer)
{
  _tprintf(_T("\nEnter Callback function()"));
  if (return_code == NIrtd_eOK)
  {
    // handle the data
     …
    //free the internally allocated space for data.
    NIrtd_freeTableGroup(tableGroup);
  }
   else
  {
  _tprintf(_T("Callback rc = %d"), return_code);
  }
  return NIrtd_eOK;
}

typedef  struct  _NIrtd_stTableGroup
{
  NIrtd_stTable  deletedValues;
  NIrtd_stTable  newValues;
  NIrtd_stTable  deltaValues;
}  NIrtd_stTableGroup;
typedef  struct  _NIrtd_stTable
{
  ULONG  numberofrows;
  ULONG  numberofcols;
  NIrtd_tTable table;
}  NIrtd_stTable;


// copy the update row content to temp row
  rc = NIrtd_allocateRow(&tempRow, updateTable, i);
  // Check return value and deal with errors.
  …
```

```
  // Allocate a value structure used to store a column value
  NIrtd_stValue columnValue;
  rc = NIrtd_allocateValue(&columnValue);
  // Check return value and deal with errors.
  …

  // Get the data from the table.
  rc = NIrtd_getCol(&columnValue, &tempRow, 0);
  // Check return value and deal with errors.
  …
  // release the value structure
  NIrtd_freeValue(&columnValue);

  // release the tempRow for next round
  NIrtd_freeRow(&tempRow);


// temporary structure used to hold skillset name
  NIrtd_stName tempSkillsetName;
  // allocate the name structure
  rc = NIrtd_allocateName(&tempSkillsetName);
  // check for error code and handle as require
  …
  rc = NIrtd_getName(&authInfo, NIrtd_SKLST_SKILLSET_ID, &skillsetId,
&tempSkillsetName);
  if ( rc == NIrtd_eOK )
    // Name is in tempSkillsetName.last_name
  }
  // cannot find the name in name cache
  else if (rc == NIrtd_eNOT_FOUND)
  {
    // go to the server directly to retrieve the name
    rc = NIrtd_getFailedName(&authInfo, NIrtd_SKLST_SKILLSET_ID,
&skillsetId, &tempSkillsetName);
    if (rc != NIrtd_eOK)
    {
      _tprintf(_T("\nCannot find the skillset name in server"));
      return;
    }

    // Name is in tempSkillsetName.last_name
  }
  else
  {
    _tprintf(_T("\nOther error in getting name cache : %d"), rc);
    return;
  }

  NIrtd_freeName(&tempSkillsetName);
```

```
// free name cache
  rc = NIrtd_removeNameCacheforDataColumn(&authInfo,
    NIrtd_SKLST_SKILLSET_ID);
  // stop data stream before exit
  rc = NIrtd_stopDataStream(&authInfo, requestId);

  // deallocate all structures allocated
  rc = NIrtd_freeQuery(&query);
  // perform logout
  rc = NIrtd_logout(&authInfo);

rc = NIrtd_setRecovery(90000,10000);
```

Chapter 6 - CCMS Open Queue Web Service

```
public String queueContact(String name, String phoneNumber, String
comments) throws LogInFailedFault, CreateOQContactFailedFault,
LogOffFailedFault, LogOffSessionFailedFault {
        System.out.println("Queueing a contact");
        instantiateProxy();
        SsoToken ssoToken = logon();
        IntrinsicArray intrinsics = createIntrinsics(name, phoneNumber,
comments);
        String contactId=createContactId();
        proxy.createOQContact(contactId, "OutsideSalesAgentWeb",
intrinsics, ssoToken);
        logoff(ssoToken);
        return "";
    }

private void instantiateProxy() {
        proxy=new SOAOIOpenQ().getPort(OpenQ.class);
}


private SsoToken logon() throws LogInFailedFault,
LogOffSessionFailedFault {
        SsoToken ssoToken=null;
        AuthenticationLevel authLevel=new AuthenticationLevel();
        authLevel.setDomain("open_queue");
        authLevel.setUsername("OpenWsUser");
        authLevel.setPassword("Password123");
        try {
        ssoToken=proxy.logIn(authLevel);
        } catch(Exception ex) {
            // Try to logoff and then log back in again.
            LogOffSessionRequestType parms=new
LogOffSessionRequestType();
            parms.setAuthenticationLevel(authLevel);
            proxy.logOffSession(parms);
            ssoToken=proxy.logIn(authLevel);
        }
        return ssoToken;
}
```

```
private IntrinsicArray createIntrinsics(String name, String
phoneNumber, String comments) {

        IntrinsicArray intrinsics=new IntrinsicArray();
        intrinsics.getItem().add(createIntrinsic("NAME", name));
        intrinsics.getItem().add(createIntrinsic("PHONE_NUMER",
phoneNumber));
        intrinsics.getItem().add(createIntrinsic("COMMENTS", comments));
        return intrinsics;
}

private Intrinsic createIntrinsic(String name, String value) {
        Intrinsic intrinsic=new Intrinsic();
        intrinsic.setKey(name);
        intrinsic.setValue(value);
        return intrinsic;
}

private String createContactId() {
        int timeAsInt=(int) System.currentTimeMillis() & 0xffff;
        return Integer.toString(timeAsInt);
}


private void logoff(SsoToken ssoToken) throws LogOffFailedFault {
        proxy.logOff(ssoToken);
}
```

```
private OpenQ setupConnection( String wsdlLocation ) {
  OpenQ port = null;
  URL wsdlURL = null;
  File wsdlFile = null;
  try {
  wsdlFile = new File(wsdlLocation);
  if (wsdlFile.exists()) {
  wsdlURL = wsdlFile.toURL();
  } else {
  wsdlURL = new URL(wsdlLocation);
  }
  } catch (MalformedURLException e) {
  e.printStackTrace();
  }
  System.out.println("OIOpenQ connecting to : " + wsdlURL.toString());
  try {
  SOAOIOpenQ ss = new SOAOIOpenQ(wsdlURL, OPENQ_SERVICE_QNAME);
  port = ss.getOpenQ();
  System.out.println("OIOpenQ connected to : " + wsdlURL.toString());
  } catch ( Exception ex ) {
  ex.printStackTrace();
  }
  return port;
}


private static SsoToken issueLogOnRequest( OpenQ port,
AuthenticationLevel auth ) {
  SsoToken sso = null;
  if ( auth != null ) {
  if ( auth.getUsername() == null ) {
  System.err.println("Username not specified");
  } else if ( auth.getPassword() == null ) {
  System.err.println("Password not specified");
  } else {
  try {
  sso = port.logIn(auth);
  } catch (LogInFailedFault ex) {
  System.err.println("Cause : " + (ex.getCause() != null ?
ex.getCause() : "unknown"));
  System.err.println("Message : " + (ex.getMessage() != null ?
ex.getMessage() : "unknown"));
  ex.printStackTrace();
  sso = null;
  }
  }
  } else {
  System.err.println("No authentication details specified");
```

```
  }
  return sso;
}


4Create a Contact
  private static Contact issueCreateContactRequest(
  OpenQ port,
  String externalContactId,
  String outOfProviderAddressName,
  HashMap<String, String> intrinsicsMap,
  SsoToken sso)
  throws CreateOQContactFailedFault {
  IntrinsicArray intrinsics = new IntrinsicArray();
  if ( intrinsicsMap != null ) {
  Iterator<String> specifiedIntrinsicKeysIter =
intrinsicsMap.keySet().iterator();
  while ( specifiedIntrinsicKeysIter.hasNext() ) {
  String intrinsicKeyStr = specifiedIntrinsicKeysIter.next();
  String intrinsicValueStr = intrinsicsMap.get(intrinsicKeyStr);

  Intrinsic intrinsic = WsFactoryManager.wsbt().createIntrinsic();
  intrinsic.setKey(intrinsicKeyStr);
  intrinsic.setValue(intrinsicValueStr);
  intrinsic.setImmutable(true);

  intrinsics.getItem().add(intrinsic);
  }
  }
  Holder<Contact> contactHolder = new Holder<Contact>();
  contactHolder.value = WsFactoryManager.wsbt().createContact();
  return port.createOQContact(
  externalContactId,
  outOfProviderAddressName,
  intrinsics,
  sso);
}



private static Contact issueGetContactRequest(
  OpenQ port,
  String externalContactId,
  SsoToken sso)
  throws GetOQContactFailedFault {
  return port.getOQContact(externalContactId, sso);
}
```

```
private static boolean issueDropContactRequest(
  OpenQ port,
  String contactId,
  SsoToken sso) throws DropOQContactFailedFault {
  return port.dropOQContact(contactId, sso );
}

private static boolean issueLogOffRequest( OpenQ port, SsoToken sso ) {
  boolean isLogOffSuccessful = false;
  if ( sso != null ) {
  try {
  isLogOffSuccessful = port.logOff(sso);
  } catch (LogOffFailedFault ex) {
  ex.printStackTrace();
  sso = null;
  }
  } else {
  System.err.println("No SSO specified");
  }
  return isLogOffSuccessful;
}
```

```
public String reserveLandingPad(String destinationCDN, String
contactData)
        throws Exception {
        SOAOIOpenNetworking service = new SOAOIOpenNetworking();
        OpenNetworking proxy = service.getPort(OpenNetworking.class);
        ReserveCDNLandingPadRequestType parameters =
            new ReserveCDNLandingPadRequestType();
        Authentication authentication = buildAuthentication();
        parameters.setAuthentication(authentication);
        parameters.setContactData(contactData);
        parameters.setContactGUID(createNewContactId());
        parameters.setDestinationCDN(destinationCDN);
        parameters.setIntrinsics(new IntrinsicArray());
        parameters.setReasonCode(OpenNetReason.TRANSFER_OPEN_NET_INIT);
        ReserveLandingPadResponseType resp =
proxy.reserveCDNLandingPad(parameters);
        //User interface should tell user to do the transfer after we
return.
        return resp.getLandingPadCDN().getName();
}


public void completeTransfer(String landingPadCDN)
        throws LandingPadCancellationFault {
        SOAOIOpenNetworking service = new SOAOIOpenNetworking();
        OpenNetworking proxy = service.getPort(OpenNetworking.class);
        Authentication authentication=buildAuthentication();
        // Release the landing pad
        CancelCDNLandingPadRequestType cancelParameters =
            new CancelCDNLandingPadRequestType();
        cancelParameters.setAuthentication(authentication);
        cancelParameters.setLandingPadCDN(landingPadCDN);
        proxy.cancelCDNLandingPad(cancelParameters);
}


private Authentication buildAuthentication() {
        Authentication authentication = new Authentication();
        authentication.setUsername("OpenWsUser");
        authentication.setPassword("Password123");
        authentication.setDomain("open_networking");
        return authentication;
}
```

```
private ContactGUID createNewContactId() {
        ContactGUID contactGUID = new ContactGUID();
        contactGUID.setGuid(Integer.toString((int)
(System.currentTimeMillis() & 0xffff)));
        return contactGUID;
}
```

Chapter 9 - CCMS Transfer to Landing Pad Use Case

```
package com.nortel.soa.oi.opennetworking;
//..import statements not shown for brevity…
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.4-b01
 * Generated source version: 2.2
 *
 */
@WebService(name = "OpenNetworking", targetNamespace =
"http://www.nortel.com/soa/oi/OpenNetworking")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
@XmlSeeAlso({
    com.nortel.soa.oi.cct.types.ObjectFactory.class,
    com.nortel.soa.oi.opennetworking.types.ObjectFactory.class,

org.oasis_open.docs.wsrf._2004._06.wsrf_ws_basefaults_1_2_draft_01.Obje
ctFactory.class,
    org.xmlsoap.schemas.ws._2003._03.addressing.ObjectFactory.class
})

public interface OpenNetworking {
    /**
     *
     * @param parameters
     * @return
     *     returns
com.nortel.soa.oi.opennetworking.types.ReserveLandingPadResponseType
     * @throws LandingPadReservationFault
     */
    @WebMethod(operationName = "ReserveCDNLandingPad", action =
"http://www.nortel.com/soa/oi/OpenNetworking/ReserveCDNLandingPad")
    @WebResult(name = "ReserveLandingPadResponse", targetNamespace =
"http://www.nortel.com/soa/oi/OpenNetworking/types", partName =
"parameters")
    public ReserveLandingPadResponseType reserveCDNLandingPad(
        @WebParam(name = "ReserveCDNLandingPadRequest", targetNamespace
= "http://www.nortel.com/soa/oi/OpenNetworking/types", partName =
"parameters")
        ReserveCDNLandingPadRequestType parameters)
        throws LandingPadReservationFault
    ;
    // Other methods not shown for brevity.
}
```

```
public class ReserveLandingPadTest {
  private static final String
OI_WSDL="http://aaccedsremote:9080/SOAOI/services/OpenNetworking?wsdl";
  private static final String
BAD_OI_WSDL="http://aaccedsremote:9080/SOAOI/services/OpenNetwrking?
wsdl";

  @Test
  public void testGetProxy() throws Exception {
  URL wsdlUrl=new URL(OI_WSDL);

  SOAOIOpenNetworking service=new SOAOIOpenNetworking(wsdlUrl);
  OpenNetworking proxy=service.getPort(OpenNetworking.class);

  }

  @Test(expected=WebServiceException.class)
  public void testBadGetProxy() throws Exception {
  URL wsdlUrl=new URL(BAD_OI_WSDL);

  SOAOIOpenNetworking service=new SOAOIOpenNetworking(wsdlUrl);
  OpenNetworking proxy=service.getPort(OpenNetworking.class);

  }
}


@Test
  public void testReserveLandingPad() throws Exception {
  URL wsdlUrl=new URL(OI_WSDL);

  SOAOIOpenNetworking service=new SOAOIOpenNetworking(wsdlUrl);
  OpenNetworking proxy=service.getPort(OpenNetworking.class);

  ReserveCDNLandingPadRequestType parameters=new
ReserveCDNLandingPadRequestType();
  Authentication authentication=new Authentication();
  authentication.setUsername("OpenWsUser");
  authentication.setPassword("Password123");
  authentication.setDomain("open_networking");
  parameters.setAuthentication(authentication);

  parameters.setContactData("landingPadAttachedData");

  ContactGUID contactGUID=new ContactGUID();
  contactGUID.setGuid("12667");
  parameters.setContactGUID(contactGUID);

  parameters.setDestinationCDN("1501");
  parameters.setIntrinsics(new IntrinsicArray());
  parameters.setReasonCode(OpenNetReason.TRANSFER_OPEN_NET_INIT);
```

```
  PropertyPrinter.printDetails(parameters);
  ReserveLandingPadResponseType
response=proxy.reserveCDNLandingPad(parameters);
  PropertyPrinter.printDetails(response);
  PropertyPrinter.printDetails(response.getLandingPadCDN());
}



package com.avaya.demo;
import java.beans.BeanInfo;
import java.beans.Introspector;
import java.beans.PropertyDescriptor;
import java.lang.reflect.Method;

public class PropertyPrinter {
  public static void printDetails(Object o) {
  try {
  BeanInfo info = Introspector.getBeanInfo(o.getClass());
  for (PropertyDescriptor pd: info.getPropertyDescriptors()) {
  Method readMethod=pd.getReadMethod();
  if (readMethod != null) {
  Object value=readMethod.invoke(o, new Object[0]);
  System.out.printf("  %s=%s\n", pd.getName(), value);
  }
  }
  } catch (Exception e) {
  e.printStackTrace();
  }
  }
}


//In real life, we'd execute the transfer here.

  // Release the landing pad
  CancelCDNLandingPadRequestType cancelParameters=new
CancelCDNLandingPadRequestType();
  cancelParameters.setAuthentication(authentication);

cancelParameters.setLandingPadCDN(response.getLandingPadCDN().getName()
);
  System.out.println("Cancel parameters...");
  PropertyPrinter.printDetails(cancelParameters);
  CancelLandingPadResponseType
cancelResponse=proxy.cancelCDNLandingPad(cancelParameters);
  System.out.println("response...");
  PropertyPrinter.printDetails(cancelResponse);
```

```
interface NIDXProvider
{
  // Some definitions omitted for clarity
  NIDXProviderMessaging DX_RegisterProvider(
    in NTUINT32 providerId,
    in UserId userIdp,
    in Version versionp)
    raises(OperationFailed );
  void DX_DeRegisterProvider(
    in NIDXProviderMessaging providerMsgObj)
  raises (OperationFailed);
  void DX_KeepAliveProvider(
    in NTUINT32 providerId)
  raises(OperationFailed );
};


interface NIDXProviderMessaging
{
  // Some definitions omitted for clarity
  void DX_GetMessage(
    in DxOperationMode opMode,
    out HDXMessage message)
  raises (OperationFailed);
  void DX_MessageResponse(
    in HDXMessage message)
  raises (OperationFailed);
};

struct HDXMessage
{
  NTUINT32 MsgReferenceId;
  NTUINT32 MsgProviderId;
  NTUINT32 MsgApplicationId;
  NTUINT32 MsgType;
  DxCallId HdxCallId;
  NTUINT32 Time;
  InfoList Info;
  NTUINT32 Reserved[10];
};

typedef unsigned short NTCHAR;
typedef unsigned short NTUINT16;
typedef unsigned long  NTUINT32;
const NTUINT32 NI_ELEMENT_SEQ = 10;
const NTUINT32 NI_INFO_SIZE = 80;
typedef sequence<NTCHAR, NI_INFO_SIZE> InfoCell;
typedef sequence<InfoCell, NI_ELEMENT_SEQ> InfoList;
```

```
idlj -td src NIDXMessage.idl
idlj -td src NIDXProvider.idl


private NIDXProvider findProviderInterface()
  throws InvalidName, CannotProceed, NotFound,   InvalidName {
   String[] orbArgs = {"-ORBInitRef",
            "NameService=corbaloc:iiop:" +
            nameServiceHost + ":" + nameServicePort
            + "/NameService"};
  ORB orb = ORB.init(orbArgs, null);
  org.omg.CORBA.Object obj =
      orb.resolve_initial_references("NameService");
  NamingContextExt ctx =
    NamingContextExtHelper.narrow(obj);
    NIDXProvider providerInterface =

NIDXProviderHelper.narrow(ctx.resolve_str("NortelNetworks/SymposiumCall
CenterServer/HDX"));
    return providerInterface;
}

@Test
public void testReceiveSyncMessage() throws Exception {
  NIDXProvider providerInterface = findProviderInterface();
  UserId uid = new
  UserId(CORBATools.string2short("nortel",
   MaxIdLen.value),
                 CORBATools.string2short("nortel",
    MaxIdLen.value));
  Version version = new Version(HDX_MajVersion.value,
   HDX_KeepAliveMinVersion.value);
  NIDXProviderMessaging messaging =
    providerInterface.DX_RegisterProvider(2106,
    uid, version);
  HDXMessageHolder messageHolder = new HDXMessageHolder();
    // cont'd …

// See code on previous page…
  log.info("Waiting for message from AACC...");
  try {
  messaging.DX_GetMessage(DxOperationMode.DXOM_SYNC,
  messageHolder);
  } catch (OperationFailed ex) {
  String reason = reasonToString(ex.reason);
  log.log(Level.SEVERE,
    "Operation failed: reason="
    + reason, ex);
  throw ex;
  }
```

```
  // cont'd…

// See code above…
  logMessage(Level.INFO, messageHolder.value);
  if (messageHolder.value.MsgType ==
    DxMessageType._DXMT_ReqRespMsg) {
  /*
  * We'll go ahead and use the same message,
    * so as to preserve the
  * header info.
  */
            log.info("Replying to message.");
            HDXMessage message = messageHolder.value;
            message.MsgType = DxMessageType._DXMT_RespMsg;
            message.Info =
     InfoAdapter.toInfo(new String[]{"Sample output"},
     10, 80);
            messaging.DX_MessageResponse(message);
        }
}


String reasonToString(short reason) {
        if (reason == ExceptionReason._CannotCreateMessage) {
            return "Cannot create message";
        }
        if (reason == ExceptionReason._InvalidObject) {
            return "Invalid Object";
        }
        if (reason == ExceptionReason._InvalidOperationMode) {
            return "Invalid operation mode";
        }
        if (reason == ExceptionReason._InvalidResponseMessage) {
            return "Invalid response message";
        }
        if (reason == ExceptionReason._InvalidTargetId) {
            return "Invalid target ID";
        }
        if (reason == ExceptionReason._NoMessage) {
            return "No message";
        }
        return "Unknown reason " + reason;
}


private void logMessage(Level level, HDXMessage message) {
        String output = "Received message:"
                + "\n  MessageReferenceId:" + message.MsgReferenceId
                + "\n  MessageProviderId:" + message.MsgProviderId
                + "\n  MessageApplicationId:" + message.MsgApplicationId
                + "\n  MessageType:" + message.MsgType
                + "\n  Time:" + message.Time;
```

```
        List<String> parameters = InfoAdapter.toStrings(message.Info);
        output = output + "\n" + "Parameters: " + parameters;
        log.log(level, output);
}


public static List<String> toStrings(short[][] input) {
        List<String> strings = new ArrayList<String>();
        for (short[] item : input) {
            makeAndAppendString(item, strings);
        }
        return strings;
}

private static void makeAndAppendString(short[] item, List<String>
strings) {
        StringBuilder sb = new StringBuilder();
        int i = 0;
        while (i < item.length && item[i] != 0) {
            sb.append((char) item[i]);
            i++;
        }
        strings.add(sb.toString());
    }
    static short[][] toInfo(String[] input, int n, int len) {
        short[][] info = new short[n][len];
        for (int i = 0; i < input.length; i++) {
            writeStringToShortArray(info[i], input[i]);
        }
        return info;
    }
    private static void writeStringToShortArray(short[] item, String
input) {
        for (int j = 0; j < item.length; j++) {
            item[j] = (j < input.length())
                    ? (short) input.charAt(j) : 0;
        }
    }
}
```

```
    SsoToken ssoToken = null;
    SOAOICCTUserService userService = null;
    UserService userServiceProxy = null;

public void logon(String userID, String password)
        throws LogInToCCTServerException {
        /* Login. */
        userService = new SOAOICCTUserService();
        userServiceProxy = userService.getPort(UserService.class);
        LogInToCCTServerRequest req = new LogInToCCTServerRequest();
        AuthenticationLevel authLevel = new AuthenticationLevel();
        authLevel.setDomain("edsremote");
        authLevel.setPassword(password);
        authLevel.setUsername(userID);
        req.setAuthenticationLevel(authLevel);
        ssoToken = userServiceProxy.logInToCCTServer(req).getSsoToken();
        System.out.println("ssoToken is " + ssoToken.getToken());
}

public void logoff()
        throws LogOffFromCCTServerException, SessionNotCreatedException
{
        /* Logoff. */
        System.out.printf("Attempting logoff with ssoToken='%s' \n",
                ssoToken.getToken());
        LogOffFromCCTServerRequest req = new
LogOffFromCCTServerRequest();
        req.setSsoToken(ssoToken);
        userServiceProxy.logOffFromCCTServer(req);
}
```

```
@WebService(name = "NotificationConsumer", targetNamespace =
"http://www.nortel.com/soa/oi/cct/NotificationConsumer")

public interface NotificationConsumer {
    /**
     *
     * @param notificationMessage
     */
    @WebMethod(operationName = "Notify", action =
"http://www.nortel.com/soa/oi/cct/notificationconsumer/service/Notify")
    @Oneway
    @RequestWrapper(localName = "Notify", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/NotificationConsumer",
className = "com.nortel.soa.oi.cct.types.notificationconsumer.Notify")
    public void notify(
        @WebParam(name = "NotificationMessage", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/NotificationConsumer")
        List<NotificationMessageHolderType> notificationMessage);
}


@WebService(

endpointInterface="com.nortel.soa.oi.cct.notificationconsumer.Notificat
ionConsumer")
 public class MyPrivateListener implements NotificationConsumer {
     @Override
     public void notify(List<NotificationMessageHolderType>
notificationMessage) {
         for(NotificationMessageHolderType note: notificationMessage) {
          Utils.printNonNullProperties(5, "   ", note);
         }
     }
 }

NotificationConsumer listener = new MyListener();
        try {
            /* Publish the endpoint. */
            listenerEndpoint = Endpoint.create(listener);
            /* Get the ip address. */
            String ipAddress = Utils.findIpPublicAddress();
            String url = String.format("http://
%s/NotificationListener", ipAddress);
            listenerEndpoint.publish(url);
            System.out.println("Published at " + url);
            // More follows…
```

```
            // See code on previous page…
            /* Login. */
            userService = new SOAOICCTUserService();
            userServiceProxy = userService.getPort(UserService.class);
            req = new LogInToCCTServerRequest();
            authLevel = new AuthenticationLevel();
            authLevel.setDomain("yourdomain");
            authLevel.setPassword("yourpassword");
            authLevel.setUsername("yourusername");
            req.setAuthenticationLevel(authLevel);
            ssoToken =
userServiceProxy.logInToCCTServer(req).getSsoToken();
            System.out.println("ssoToken is " + ssoToken.getToken());


 /* Register for events. */
            System.out.println("Subscribing...");
            String subscribeResp = userServiceProxy.subscribe(url,
ssoToken);
            System.out.println("Subscription Response:" +
subscribeResp);
            System.out.println("ssoToken is " + ssoToken.getToken());


            /* Unsubscribe */
            System.out.println("Unsubscribing...");
            userServiceProxy.unsubscribe(subscribeResp, ssoToken);
            /* Unpublish */
            System.out.println("stopping endpoint...");
            listenerEndpoint.stop();
        } catch (Exception ex) {

Logger.getLogger(CCTEventsDemo.class.getName()).log(Level.SEVERE, null,
ex);
        }
        /* Exit */
        System.out.println("Exiting...");
        System.exit(0);



     <h:body>
        <h:form>
            <ace:panel header="PizzaParadise and BurritoVille">
                <h:messages globalOnly="true"/>
                <h:panelGrid columns="3">
                    <h:panelGroup>
                        UserId:<h:inputText value="#{mainForm.userID}"
    disabled="#{mainForm.loggedOn}"/>
                    </h:panelGroup>
                    <h:panelGroup>
                        Password:<h:inputSecret
```

```
value="#{mainForm.password}"
    disabled="#{mainForm.loggedOn}"/>
                    </h:panelGroup>
                    <h:panelGroup><ace:pushButton
disabled="#{mainForm.loggedOn}"
    value="Logon" action="#{mainForm.logon()}"/>
                        <ace:pushButton disabled="#{!
mainForm.loggedOn}"
    value="Logoff" action="#{mainForm.logoff()}"/>
                    </h:panelGroup>
                </h:panelGrid>
                Primary terminal #{mainForm.terminalName}<br/>
                State: #{mainForm.connectionState}<br/>
                Brand:#{mainForm.brand}<br/>
            </ace:panel>
        </h:form>
    </h:body>
```

```
@Inject
    CCTLiaison cctLiaison;
    public String logon() {
        try {
            cctLiaison.logon(userID, password);
            /* Find our voice address. */
            terminalName = findVoiceTerminalName();
            loggedOn = true;
        } catch (Throwable ex) {
            ex.printStackTrace();
            FacesMessage message = new FacesMessage(ex.getMessage());
            FacesContext.getCurrentInstance().addMessage(null, message);
        }
        return "";
    }


private String findVoiceTerminalName() throws GetAddressesException,
SessionNotCreatedException, GetTerminalsException,
GetTerminalsException {
        for (Terminal term : cctLiaison.getAssociatedTerminals()) {
            if (!term.getTerminalName().contains("CCMM")) {
                return term.getTerminalName();
            }
        }
        return "U/K";
}
```

```
private PortableRenderer renderer = null;
    @PostConstruct
    public void init() {
        PushRenderer.addCurrentSession(PUSH_GROUP);
        renderer = PushRenderer.getPortableRenderer();
        cctLiaison.addListener(this);
}


public void logon(String userID, String password) throws
      LogInToCCTServerException,
      SessionNotCreatedException, SubscribeFailedFault {
        /* Login. */
        LogInToCCTServerRequest req = new LogInToCCTServerRequest();
        AuthenticationLevel authLevel = new AuthenticationLevel();
        authLevel.setDomain("edsremote");
        authLevel.setPassword(password);
        authLevel.setUsername(userID);
        System.out.printf("Attempting logon with userid='%s' and
password='%s'\n", userID, password);
        req.setAuthenticationLevel(authLevel);
        ssoToken =
getUserServiceProxy().logInToCCTServer(req).getSsoToken();
        System.out.println("ssoToken is " + ssoToken.getToken());
        subscribeToEvents();
}


transient UserService userServiceProxy = null;
    public UserService getUserServiceProxy() {
        if (userServiceProxy == null) {
            SOAOICCTUserService userService = null;
            userService = new SOAOICCTUserService();
            userServiceProxy = userService.getPort(UserService.class);
        }
        return userServiceProxy;
}


public void subscribeToEvents() throws
        SessionNotCreatedException, SubscribeFailedFault {
        /* Endpoint is published as part of the web application */
        String url = LISTENER_URL;
        System.out.append("Notification consumer's url is " + url);
        /* Register for events. */
        System.out.println("Subscribing...");
        subscriptionRef = getUserServiceProxy().subscribe(url,
ssoToken);
        /* Plug into dispatcher */
        String subscriptionId =
            Utils.extractSubscriptionID(subscriptionRef);
        dispatcher.addListener(subscriptionId, this);
```

```
        System.out.println("Subscription Response:" + subscriptionRef);
}


public static String extractSubscriptionID(String subscriptionRef) {
        /* Subscription ref looks like:
         *

http://myhost:8080/CCTBusinessScenario/SOAOICCT_NotificationConsumer:2e
3a7b0e-5fa2-4164-bd5c-6a4630f6b3c
         * Subscription id that comes with the events is the part after
the last ':'
         */
        return subscriptionRef.substring(
            subscriptionRef.lastIndexOf(":") +1);
}


@WebService(serviceName = "SOAOICCT_NotificationConsumer", portName =
"NotificationConsumer", endpointInterface =
"com.nortel.soa.oi.cct.notificationconsumer.NotificationConsumer",
targetNamespace =
"http://www.nortel.com/soa/oi/cct/NotificationConsumer", wsdlLocation =
"WEB-
INF/wsdl/NotificationConsumerImpl/aaccedsremote_9090/NotificationConsum
er.wsdl")
public class NotificationConsumerImpl {
    @Inject
    private NotificationDispatcher dispatcher;
    public void notify(java.util.List<NotificationMessageHolderType>
            notificationMessage) {
        dispatcher.notify(notificationMessage);
    }
}


public class NotificationDispatcher implements NotificationConsumer {
    @Override
    public synchronized void notify(List<NotificationMessageHolderType>
notificationMessage) {
        for (NotificationMessageHolderType holder :
notificationMessage) {
            String subscriptionId = holder.getSubscriptionId();
            NotificationConsumer listener =
listeners.get(subscriptionId);
            if (listener != null) {
                List<NotificationMessageHolderType> dispatchedMessages
=
                    new ArrayList<NotificationMessageHolderType>();
                dispatchedMessages.add(holder);
                /* Old listeners might hang around.  Can't let them
disrupt the
```

```
                 * current listeners, so notify in a 'try-catch' block.
                 */
                try {
                    listener.notify(dispatchedMessages);
                } catch (Throwable t) {
                    System.out.println("Caught exception " + t + " in
notify(...)");
                }
            }
        }
}


@Override
public void notify(List<NotificationMessageHolderType>
notificationMessage) {
        for (NotificationListener l : listeners) {
            l.notify(this, notificationMessage);
        }
}


@Override
public void notify(CCTLiaison liaison,
List<NotificationMessageHolderType> notificationMessage) {
        for (NotificationMessageHolderType holder :
notificationMessage) {
            if (holder.getMessage().getEventType() ==
                    EventType.TERMINAL_CONNECTION_STATE) {
                TerminalConnectionStateEventType ev =

holder.getMessage().getTerminalConnectionStateEvent();
                if
(ev.getTerminal().getTerminalName().equals(terminalName)) {
                    connectionState = ev.getNewState().toString();
                    currentContact = ev.getContact();
                    if (ev.getNewState() ==
TerminalConnectionState.RINGING) {
                        updateBrand(liaison);
                    }
                    if (ev.getNewState() ==
TerminalConnectionState.DROPPED) {
                        brand="Unknown";
                    }
                    renderer.render(PUSH_GROUP);
                }
            }
        }
}
```

```
private void updateBrand(CCTLiaison liaison) {
        try {
            String calledAddress =
                liaison.findCalledAddress(currentContact);
            if (calledAddress.equals(PIZZA_PARADISE_ADDRESS)) {
                brand = "PizzaParadise";
            } else if (calledAddress.equals(BURRITOVILLE_ADDRESS)) {
                brand = "BurritoVille";
            } else {
                brand = "Unknown";
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
}


public String findCalledAddress(Contact contact)
            throws GetCalledAddressException,
                   SessionNotCreatedException {
        ContactRequest req = new ContactRequest();
        req.setContact(contact);
        req.setSsoToken(ssoToken);

        AddressResponse resp =
            getContactServiceProxy().getCalledAddress(req);
        Address address = resp.getAddress();
        String addressName = address.getAddressName();
        return addressName;
}
```

```
package com.nortel.soa.oi.cct.userservice;
import javax.jws.WebMethod;
…
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.4-b01
 * Generated source version: 2.2
 *
 */
@WebService(name = "UserService", targetNamespace =
"http://www.nortel.com/soa/oi/cct/UserService")
…
public interface UserService {
…
 /**
     *
     * @param parameters
     * @return
     *     returns
com.nortel.soa.oi.cct.types.sessionservice.LogInToCCTServerResponse
     * @throws LogInToCCTServerException
     */
    @WebMethod(operationName = "LogInToCCTServer", action =
"http://www.nortel.com/soa/oi/cct/UserService/LogInToCCTServer")
    @WebResult(name = "LogInToCCTServerResponse", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"result")
    @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
    public LogInToCCTServerResponse logInToCCTServer(
        @WebParam(name = "LogInToCCTServerRequest", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"parameters")
        LogInToCCTServerRequest parameters)
        throws LogInToCCTServerException
    ;
}
```

```
public class CCTLoginTest {
  @Test
  public void testGetProxy() throws Exception {
  URL wsdlURL=new
URL("http://aaccedsremote:9084/SOAOICCT/services/UserService?wsdl");
  SOAOICCTUserService serviceLocator=
  new SOAOICCTUserService(wsdlURL);
  UserService userService=serviceLocator.getPort(UserService.class);
  }
}
```

```
@WebMethod(operationName = "LogInToCCTServer", action =
"http://www.nortel.com/soa/oi/cct/UserService/LogInToCCTServer")
    @WebResult(name = "LogInToCCTServerResponse", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"result")
    @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
    public LogInToCCTServerResponse logInToCCTServer(
        @WebParam(name = "LogInToCCTServerRequest", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"parameters")
        LogInToCCTServerRequest parameters)
        throws LogInToCCTServerException
    ;
```

```
@Test
public void testLogin() throws Exception {
  URL wsdlURL=new
URL("http://aaccedsremote:9084/SOAOICCT/services/UserService?wsdl");
  SOAOICCTUserService serviceLocator=
  new SOAOICCTUserService(wsdlURL);
  UserService userService=serviceLocator.getPort(UserService.class);
  LogInToCCTServerRequest req=new LogInToCCTServerRequest();
  AuthenticationLevel authLevel=new AuthenticationLevel();
  authLevel.setDomain("edsremote");
  authLevel.setUsername("1116");
  authLevel.setPassword("pw!");

  req.setAuthenticationLevel(authLevel);
  LogInToCCTServerResponse resp=userService.logInToCCTServer(req);
  System.out.println("ssoToken is:" + resp.getSsoToken().getToken());

}
```

```
   @WebMethod(operationName = "GetAddresses", action =
"http://www.nortel.com/soa/oi/cct/UserService/GetAddresses")
    @WebResult(name = "GetAddressesResponse", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"result")
    @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
    public AddressList getAddresses(
        @WebParam(name = "GetAddressesRequest", targetNamespace =
"http://www.nortel.com/soa/oi/cct/types/SessionService", partName =
"parameters")
        GetAddressesRequest parameters)
        throws GetAddressesException, SessionNotCreatedException
    ;


@Test
public void listAddresses() throws Exception {
  LogInToCCTServerResponse loginResponse=login();
  GetAddressesRequest req=new GetAddressesRequest();
  req.setSsoToken(loginResponse.getSsoToken());
  AddressList resp=userService.getAddresses(req);
  System.out.println(resp.getAddresses());
}

package com.avaya.demo;
import java.beans.BeanInfo;
import java.beans.Introspector;
import java.beans.PropertyDescriptor;
import java.lang.reflect.Method;
public class PropertyPrinter {
    public static void printDetails(Object o) {
        try {
            BeanInfo info = Introspector.getBeanInfo(o.getClass());
            for (PropertyDescriptor pd: info.getPropertyDescriptors()) {
                Method readMethod=pd.getReadMethod();
                if (readMethod != null) {
                    Object value=readMethod.invoke(o, new Object[0]);
                    System.out.printf("  %s=%s\n", pd.getName(), value);
                }
            }
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}


@Test
public void listAddresses() throws Exception {
  LogInToCCTServerResponse loginResponse=login();
  GetAddressesRequest req=new GetAddressesRequest();
```

```
  req.setSsoToken(loginResponse.getSsoToken());
  AddressList resp=userService.getAddresses(req);
  for (Address addr : resp.getAddresses()) {
  PropertyPrinter.printDetails(addr);
  }
}
```

```
Toolkit toolkit = new Toolkit();
    toolkit.Credentials =
      new CCTCredentials("1116", "edsremote", "pw");
    toolkit.Server = "aaccedsremote";
    ISession session=toolkit.Connect();


[TestMethod]
public void TestTerminals()
        {
            Login();
            ITerminal[] terminals = session.Terminals;
            System.Console.WriteLine("Printing terminals ("
                + terminals.Length + " found)");
            foreach (ITerminal terminal in terminals)
            {
                System.Console.WriteLine("Terminal "
                    + terminal.Name + ", type=" + terminal.GetType());
                System.Console.WriteLine("..addresses are " +
                    Format(terminal.RelatedAddresses));
            }
            System.Console.WriteLine();
            Logout();
}

Printing terminals (2 found)
Terminal DefaultNode_CCMM_1116,
type=Nortel.CCT.ProxyLayer.AgentTerminalProxy
..addresses are [DefaultNode_CCMM_1116]
Terminal Line 96.0.1.6, type=Nortel.CCT.ProxyLayer.AgentTerminalProxy
..addresses are [4506, 4006]


Toolkit toolkit = new Toolkit();
ISession session = null;
private void Login()
        {
            toolkit.Credentials = new CCTCredentials("1116",
                "edsremote", "pw");
            toolkit.Server = "aaccedsremote";
            session = toolkit.Connect();
}
private void Logout()
        {
            toolkit.Disconnect();
}
```

```
private String Format(IAddress[] addresses)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("[");
            Boolean first = true;
            foreach (IAddress address in addresses)
            {
                if (first)
                {
                    first = false;
                }
                else
                {
                    sb.Append(", ");
                }
                sb.Append(address.Name);
            }
            sb.Append("]");
            return sb.ToString();
}


[TestMethod]
public void TestContactEvent()
        {
            Login();
            ContactScopeEventHandler handler =
    toolkit.CreateEventHandler(new
ContactScopeEventHandler(ContactScopeEnter));
            session.ContactEnteringScope += handler;
            // Delay for 60s to catch some events
            System.Threading.Thread.Sleep(60000);
            Logout();
        }
        private void ContactScopeEnter(ContactScopeEventArgs args)
        {
            System.Console.WriteLine("Contact {0} has entered scope.",
args.Contact.ID);
}


SessionManager sessionManager = null;
private void Login()
        {
            sessionManager = new SessionManager();
            sessionManager.CCTServerName = "aaccedsremote";
            sessionManager.CCTUserDomain = "edsremote";
            sessionManager.CCTUserName = "1116";
            sessionManager.CCTUserPassword = "pw!";
            sessionManager.CommandMode =
LiteCommandExecutionMode.Synchronous;
            sessionManager
```

```
    .LogonToCCTServer(LiteAuthenticationLevel.SpecificWindowsUser);
        }
        private void Logout()
        {
            sessionManager.LogoffFromCCTServer();
}


[TestMethod]
public void TestAddresses()
        {
            Login();
            LiteAddress[] addresses =
sessionManager.GetCurrentAddressInfo();
            System.Console.WriteLine("Printing addresses ("
                + addresses.Length + " found)");
            foreach( LiteAddress address in addresses) {
                System.Console.WriteLine("Address "
                    + address.Name);
                System.Console.WriteLine("  terminals: "
                    + Format(address.RelatedTerminals));
            }
            System.Console.WriteLine();
            Logout();
}



string userID = "";
string password = "";
private void btnLogon_Click(object sender, EventArgs e)
        {
            btnLogoff.Enabled = false;
            btnLogon.Enabled = false;
            txtUserID.Enabled = false;
            txtPassword.Enabled = false;
            lblStatus.Text = "Logging on...";
            userID = txtUserID.Text;
            password = txtPassword.Text;
            BackgroundWorker loginWorker = new BackgroundWorker();
            loginWorker.DoWork += new DoWorkEventHandler(logon);
            loginWorker.RunWorkerCompleted +=new
    RunWorkerCompletedEventHandler(logonComplete);
            loginWorker.RunWorkerAsync();
}
```

```
        Toolkit toolkit = new Toolkit();
        ISession session = null;
        void logon(Object sender, DoWorkEventArgs args)
        {
            toolkit.Credentials =
    new CCTCredentials(userID, "edsremote", password);
            toolkit.Server = "aaccedsremote";
            try
            {
                session = null;
                session = toolkit.Connect();
                SetupEvents();
                args.Result = "";
            }
            catch (Exception ex)
            {
                args.Result = ex.Message;
            }
        }


void logonComplete(Object sender, RunWorkerCompletedEventArgs args)
        {
            if (session == null)
            {
                lblStatus.Text = (string)args.Result;
                btnLogon.Enabled = true;
                txtUserID.Enabled = true;
                txtPassword.Enabled = true;
            }
            else
            {
                lblStatus.Text = "Logged on.";
                btnLogoff.Enabled = true;
            }
}


private void SetupEvents()
        {
            ITerminal terminal = FindVoiceTerminal();
            SetupTerminalEvents(terminal);
        }
        private ITerminal FindVoiceTerminal()
        {
            // Pick the one that doesn't include CCMM in its name
            foreach (ITerminal terminal in session.Terminals)
            {
                if (!terminal.Name.Contains("CCMM"))
                {
                    return terminal;
                }
```

```
            }
            return null;
}


private void SetupTerminalEvents(ITerminal terminal)
        {

            TermConnStateEventHandler handler =
                toolkit.CreateGUIEventHandler(
                    new TermConnStateEventHandler(TerminalStateChange),
                    this);
            terminal.TermConnStateChanged += handler;
}


public void TerminalStateChange(TermConnStateEventArgs args)
        {
            if (args.NewState == TerminalConnectionState.Ringing)
            {
                IContact contact = args.Contact;
                if (contact.CalledAddress == PIZZA_PARADISE_ADDRESS)
                {
                    lblCallTarget.Text = "Call for PizzaParadise";
                    // Load menus, etc for PizzaParadise
                }
                else if (contact.CalledAddress == BURRITOVILLE_ADDRESS)
                {
                    lblCallTarget.Text = "Call for BurritoVille";
                    // Load menus, etc for BurritoVille
                }
                else
                {
                    lblCallTarget.Text = "Call for unknown address";
                    // Take appropriate action…
                }
            }
            else if (args.NewState == TerminalConnectionState.Dropped)
            {
                lblCallTarget.Text = "Waiting for call..." +
args.NewState.ToString();
            }
}
```

```
private void btnPlaceOrder_Click(object sender, EventArgs e)
        {
            // Not shown - Interact with fulfillment system to place
the order...
            // Drop the contact.
            BackgroundWorker worker = new BackgroundWorker();
            worker.DoWork+=new DoWorkEventHandler(DropContact);
            worker.RunWorkerAsync();
        }
        private void DropContact(object sender, EventArgs e)
        {
            FindVoiceTerminal()
                .TerminalConnections[0].Connection.Disconnect();
        }
}
```

Chapter 15 - CCT Hot Desking

```
Toolkit toolkit = new Toolkit();
    toolkit.Credentials =
    new CCTCredentials("1116", "edsremote", "pw");
    toolkit.Server = "aaccedsremote";
    ISession session=toolkit.Connect();

  Toolkit toolkit = new Toolkit();
    toolkit.Credentials =
    new CCTCredentials("1116", "edsremote", "pw");
    toolkit.Server = "aaccedsremote";
  toolkit.Workstation = Environment.MachineName;
    ISession session=toolkit.Connect();


SsoToken ssoToken=login();

  TerminalList resp = listTerminals(ssoToken);
  Terminal terminalOfInterest=
  chooseTerminalFromList(resp.getTerminals());
  /* Now login to the terminal of interest. */
  AgentTerminalLoginRequest req=new
    AgentTerminalLoginRequest();
  req.setAgentTerminal(terminalOfInterest);
  req.setAgentId("1116");
  req.setSsoToken(ssoToken);
  req.setPassword("1116");
  req.setInitialState(AgentReadyStatus.READY);
  try {
  agentTerminalService.login(req);
  } catch(Throwable t) {
  t.printStackTrace();
  throw t;
  }


URL userServiceWsdlURL=new
  URL("http://aaccedsremote:9084/SOAOICCT/services/UserService?wsdl");
SOAOICCTUserService serviceLocator=
  new SOAOICCTUserService(userServiceWsdlURL);
userService = serviceLocator.getPort(UserService.class);

URL agentTerminalServiceWsdlURL=new
  URL("http://aaccedsremote:9084/SOAOICCT/services/AgentTerminalService
?wsdl");
SOAOICCTAgentTerminalService agentTerminalServiceLocator=
  new SOAOICCTAgentTerminalService(agentTerminalServiceWsdlURL);
agentTerminalService =
agentTerminalServiceLocator.getPort(AgentTerminalService.class);
```

```
private SsoToken login() throws MalformedURLException,
  LogInToCCTServerException {
  LogInToCCTServerRequest req=new LogInToCCTServerRequest();
  AuthenticationLevel authLevel=new AuthenticationLevel();
  authLevel.setDomain("edsremote");
  authLevel.setUsername("1116");
  authLevel.setPassword("pw");

  req.setAuthenticationLevel(authLevel);
  LogInToCCTServerResponse resp=userService.logInToCCTServer(req);
  return resp.getSsoToken();
}


private TerminalList listTerminals(SsoToken ssoToken)
  throws GetTerminalsException, SessionNotCreatedException {
  GetTerminalsRequest req=new GetTerminalsRequest();
  req.setSsoToken(ssoToken);
  TerminalList resp=userService.getTerminals(req);
  return resp;
}


private Terminal chooseTerminalFromList(List<Terminal> list) {
  Terminal terminalOfInterest=null;

  for (Terminal term : list) {
  if("Line 96.0.1.6".equals(term.getTerminalName())) {
  terminalOfInterest=term;
  break;
  }
  }
  return terminalOfInterest;
}


private TerminalList listTerminals(SsoToken ssoToken)
  throws GetTerminalsException, SessionNotCreatedException {
  GetTerminalsRequest req=new GetTerminalsRequest();
  req.setSsoToken(ssoToken);
  TerminalList resp=userService.getTerminals(req);
  return resp;
}
```

```
/* Now login to the terminal of interest. */
  AgentTerminalLoginRequest req=new
    AgentTerminalLoginRequest();
  req.setAgentTerminal(terminalOfInterest);
  req.setAgentId(agentId);
  req.setSsoToken(ssoToken);
  req.setPassword(agentPassword);
  req.setInitialState(AgentReadyStatus.READY);
  try {
  agentTerminalService.login(req);
  } catch(Throwable t) {
  t.printStackTrace();
  throw t;
}
```

Chapter 16 - Use Case – Communications Control Toolkit Reference Clients

```
public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            ClassPathResource res = new ClassPathResource("spring-
beans.xml");
            XmlBeanFactory beanFactory = new XmlBeanFactory(res);
            ((SOARefClient)
beanFactory.getBean("refClient")).setVisible(true);
         }
      });
}


<bean id="refClient" class="com.nortel.rc.gui.SOARefClient">
      <property name="requestHandler">
         <ref bean="requestHandler" />
      </property>
   </bean>
   <bean id="requestHandler"
class="com.nortel.rc.handler.RequestHandler">
      <constructor-arg index="0">
         <value>log4j.properties</value>
      </constructor-arg>
      <property name="notificationHandler">
         <ref bean="notificationHandler" />
      </property>
      <property name="commands">
         <map>
            <entry value-ref="createContactCommand">
               <key>
                  <value>CreateContact</value>
               </key>
            </entry>



   <bean id="command" class="com.nortel.rc.command.Command">
      <property name="refClient">
         <ref bean="refClient" />
      </property>
   </bean>
   <bean id="createContactCommand"
class="com.nortel.rc.command.CreateContactCommand"
      parent="command">
      <property name="contactDAO">
         <ref bean="contactDAO" />
```

```
      </property>
   </bean>


<bean id="contactDAO"
class="org.springframework.aop.framework.ProxyFactoryBean">
      <property name="proxyInterfaces">
         <value>com.nortel.rc.dao.IContactDAO</value>
      </property>
      <property name="target">
         <ref local="contactDAOBeanTarget" />
      </property>
      <property name="interceptorNames">
         <list>
            <value>loggingThrowsAdvisor</value>
            <value>traceLoggingInterceptor</value>
         </list>
      </property>
</bean>
…
<bean id="contactDAOBeanTarget"
class="com.nortel.rc.dao.SessionServiceDAO">
      <property name="serviceQName">
         <ref bean="cmfOiServiceQName" />
      </property>
      <property name="serviceUrl">
         <value>/SOAOICCT/services/SessionService?wsdl</value>
      </property>
</bean>



public CallResult cctLogin(String uname, String pwd, String domain)
         throws DAOException {
      LogInToCCTServerRequest request = new LogInToCCTServerRequest();
      AuthenticationLevel authenticationLevel = new
AuthenticationLevel();
      authenticationLevel.setUsername(uname);
      authenticationLevel.setPassword(pwd);
      authenticationLevel.setDomain(domain);
      request.setAuthenticationLevel(authenticationLevel);
      LogInToCCTServerResponse response = null;
      try {
         response = getPort().logInToCCTServer(request);
      } catch (Exception e) {
         throw new DAOException(e);
      }
      CallResult result = new CallResult();
      result.setMessage(response.getSsoToken().getToken());
      return result;
}
```

```
private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
      getRequestHandler().handle("AnswerContact", getContactParams());
}


private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
      getRequestHandler().handle("AnswerContact", getContactParams());
}

    <bean id="requestHandler"
class="com.nortel.rc.handler.RequestHandler">
      …
      <property name="commands">
         <map>
            …
            <entry value-ref="answerContactCommand">
               <key>
                  <value>AnswerContact</value>
               </key>
            </entry>
            …

<bean id="answerContactCommand"
class="com.nortel.rc.command.AnswerContactCommand"
      parent="command">
      <property name="contactDAO">
         <ref bean="contactDAO" />
      </property>
</bean>


public void execute(Map<String, Object> params) throws Exception {
      TerminalTO terminal = (TerminalTO) params.get("terminal");
      getContactDAO().answerContact((String) params.get("contactId"),
terminal,
            ssoToken);
}


 <bean id="contactDAOBeanTarget"
class="com.nortel.rc.dao.SessionServiceDAO">
      <property name="serviceQName">
         <ref bean="cmfOiServiceQName" />
      </property>
      <property name="serviceUrl">
         <value>/SOAOICCT/services/SessionService?wsdl</value>
      </property>
   </bean>
```

```
public CallResult answerContact(String contactId, TerminalTO term,
String sso)
         throws DAOException {
      TerminalContactRequest contactRequest = new
TerminalContactRequest();
      Terminal terminal = new Terminal();
      SsoToken ssoToken = new SsoToken();
      Contact contact = new Contact();

terminal.setTerminalType(TerminalType.valueOf(term.getTerminalType()));
      terminal.setTerminalName(term.getTerminalName());
      contact.setContactId(contactId);
      ssoToken.setToken(sso);
      contactRequest.setTerminal(terminal);
      contactRequest.setSsoToken(ssoToken);
      contactRequest.setContact(contact);
      try {
         getPort().answerContact(contactRequest);
      } catch (Exception e) {
         throw new DAOException(e);
      }
      CallResult result = new CallResult();
      return result;
}


private void MainForm_Load(object sender, System.EventArgs e)
    {
      // Hide all panels (agent, desktop and route point) until
connected to the server.
      this.Visible = false;
      EnablePanels(false, false, false);
      this.Visible = true;
      _sessMgr.Server = Persistence.Instance.Server;
      _sessMgr.Credentials = Persistence.Instance.Credentials;
      if (_sessMgr.Server == null)
      {
        this.Show();
        _sessMgr.Server = new ServerSettings();
        prefsMenu_Server_Click(this, EventArgs.Empty);
        prefsMenu_Credentials_Click(this, EventArgs.Empty);
      }
      UpdateOnlineStatus();
      if (Persistence.Instance.AutoConnect)
      {
        this.Show();
        ConnectToServer();
      }
}
```

```
private void ConnectToServer()
    {
      _callsRcvd = 0;
      ConnectionStatusForm csf = new ConnectionStatusForm(_sessMgr,
Persistence.Instance.AutoConnect);
      DialogResult result = csf.ShowDialog(this);
      csf.Dispose();
      ClearStatusMessage();
}

public void Connect()
    {
      _myToolkit.Server = _curServer.PrimaryServer;
      _myToolkit.CampusAlternateServer =
_curServer.CampusAlternateServer;
      _myToolkit.GeographicAlternateServer =
_curServer.GeographicAlternateServer;
      _myToolkit.Port = _curServer.Port;
      if (_curServer.Hotdesking)
      {
        if (_curServer.Workstation == null)
          _myToolkit.Workstation = Environment.MachineName;
        else
          _myToolkit.Workstation = _curServer.Workstation;
      }
      else
        _myToolkit.Workstation = null;
      _mySession = _myToolkit.Connect();

}


private void OnSessionConnectedEvent(SessionConnectedEventArgs e)
    {
      // Ignore the event if the session manager has been disposed
      // or the session has already been torn down.
      if (!_disposed)
      {
        _mySession = e.Session;
        // Handle the connected event on a background thread so we don't
        // hold up the toolkit thread.
        ThreadPool.QueueUserWorkItem(new
WaitCallback(HandleConnectEvent), new EventArgs());
      }
}
```

```
private void OnSessionConnectedEvent(SessionConnectedEventArgs e)
    {
      // Ignore the event if the session manager has been disposed
      // or the session has already been torn down.
      if (!_disposed)
      {
        _mySession = e.Session;
        // Handle the connected event on a background thread so we don't
        // hold up the toolkit thread.
        ThreadPool.QueueUserWorkItem(new
WaitCallback(HandleConnectEvent), new EventArgs());
      }
}


private void OnSessionConnected(object sender, EventArgs e)
    {
      this.Invoke(_sessionConnectedDelegate, sender, e);
}


private void SessionConnected(object sender, EventArgs e)
    {
      try
      {
        InitUserInterface();
      }
      catch (OperationFailureException ofe)
      {
        if (ofe.Error == Error.ServerCommunicationFailure)
        {
          LogMessage("The connection to the server dropped while
setting up the user-interface.", Color.Red);
        }
        else
        {
          string message = String.Format("A CCT operation failed while
setting up the user interface: Error={0}, Operation={1}, Text={2}",
            ofe.Error, ofe.Operation, ofe.Message);
          LogMessage(message, Color.Red);
        }
      }
      catch (Exception ex)
      {
        LogException(ex);
      }
}
```

```
public class AcceptButton : ConnectionButtonWrapper
  {
    public AcceptButton(Button button, ConnectionSelector selector) :
      base(button, selector)
    {
    }
    protected override void Execute(params object[] parms)
    {
      GiveStatusMessage("Accepting call...");
      Selection.Connection.Accept();
    }
    protected override bool EvaluateCCTState()
    {
      if (Selection.Capabilities == null)
        return false;
      else
        return Selection.Capabilities.CanAccept;
    }
}
```

Chapter 20 - CCMM Agent Web Services Use Case

```
sessionKey = utilityWS.AgentDesktopLogin(txtUsername.Text,
txtPassword.Text);
    btnLogin.Text = "Logoff";
    loggedIn = true;
    txtPassword.Enabled = false;
    txtUsername.Enabled = false;
    txtContactID.Enabled = true;
    btnReadContact.Enabled = true;
    btnNew.Enabled = true;


cmbClosedReasonCode.Items.Clear();
    foreach (AWClosedReasonCode code in
contactWS.GetAllClosedReasonCodes(sessionKey))
    {
        cmbClosedReasonCode.Items.Add(code);
    }
    if (cmbClosedReasonCode.Items.Count > 0)
        cmbClosedReasonCode.Enabled = true;
    cmbClosedReasonCode.DisplayMember = "name";
    cmbClosedReasonCode.ValueMember = "numericvalue";


 // Now call the web method - either with or without attachments
    if (txtAttach.Text != "")
        contactsCreated =
emailWS.OriginateMailWithAttachments(txtSubject.Text.Trim(),
txtBody.Text.Trim(), txtTo.Text.Trim(), txtCC.Text.Trim(),
txtBCC.Text.Trim(), txtFrom.Text.Trim(),
            txtCharSet.Text.Trim(), txtUsername.Text.Trim(), skillsetID,
            CRCid, sessionKey, AttachmentsList);
    else
        contactsCreated = emailWS.OriginateMail(txtSubject.Text.Trim(),
txtBody.Text.Trim(), txtTo.Text.Trim(), txtCC.Text.Trim(),
txtBCC.Text.Trim(), txtFrom.Text.Trim(),
            txtCharSet.Text.Trim(), txtUsername.Text.Trim(), skillsetID,
            CRCid, sessionKey);



utilityWS.AgentDesktopLogoff(txtUsername.Text);
    ClearText();
    txtContactID.Clear();
    sessionKey = "";
    contact = null;
    contactID = "0";
    loggedIn = false;
    btnLogin.Text = "Login";
```

```
txtPassword.Enabled = true;
txtUsername.Enabled = true;
ButtonsEnabled(false);
btnSend.Enabled = false;
txtUsername.Focus();
```

```
String sessionKey = "";
        long contactId = -1;
        AnonymousLoginResult anonLoginRes = null;
        try {
            /* Hypothesis. */
            /* Get anon session key. */
            anonLoginRes = utilityWsProxy.getAnonymousSessionKey();
            sessionKey = anonLoginRes.getSessionKey();
            /* Get anon customer id. */
            long customerId =
utilityWsProxy.getAnonymousCustomerID(anonLoginRes,
“somebody@somecompany.com", “555-5555");


 /* Get the skillset id. */
            long skillsetId =
skillsetWsProxy.getSkillsetByName("WC_Default_Skillset",
sessionKey).getId();


 /* Fill in contact information in a WriteContact structure. */
            CIContactWriteType newContact = new CIContactWriteType();
            newContact.setSkillsetID(skillsetId);
            /* go to CustomerWs and request text chat. */
            contactId = customerWsProxy.requestTextChat(customerId,
newContact, false, sessionKey);
            /* Returned value is the new contact. */



            boolean inSession = awaitSessionStart(contactId,
sessionKey, 60.0);
            if (inSession) {
                System.out.println("Contact was accepted.");
            } else {
                System.out.println("Session was not picked up");
            }
…
private boolean awaitSessionStart(long contactId, String sessionKey,
double seconds) throws InterruptedException {
        /* Call CIWebCommsWsSoap updateAliveTimer().  If exception, no
session.*/
        long startTime = System.currentTimeMillis();
        boolean inSession = false;
        while (System.currentTimeMillis() - startTime < seconds * 1000l
& !inSession) {
            try {
                webCommsWsProxy.updateAliveTime(contactId, sessionKey);
```

```
                inSession=true;
            } catch (Exception e) {
                if (! e.getMessage().contains("8756")) {
                    throw e;
                }
            }
            Thread.sleep(5000);
}


            long startTime = System.currentTimeMillis();
            CIDateTime lastReadTime = new CIDateTime();
            lastReadTime.setMilliseconds(0);
            while (System.currentTimeMillis() - startTime < 60000) {
                /* See if there's a message available. */
                CIMultipleChatMessageReadType response =
webCommsWsProxy.readChatMessage(contactId, lastReadTime, false,
sessionKey);
                lastReadTime = response.getLastReadTime();
                ArrayOfCIChatMessageReadType messagesHolder =
response.getListOfChatMessages();
                if (messagesHolder != null) {
                    for (CIChatMessageReadType message :
messagesHolder.getCIChatMessageReadType()) {
                      if (message.getChatMessageType() ==
CIChatMessageType.CHAT_MESSAGE_FROM_AGENT) {

System.out.println(message.getChatMessage());
                      }
                    }
                }
                /* Wait for a while before looping again. */
                Thread.sleep(5000);
            }


 try {
                sessionKey =
utilityWsProxy.customerLogin("greg.trasuk@webagesolutions.com",
"password");
            } catch (Exception e) {
                if (e.getMessage().contains("Already")) {

utilityWsProxy.customerLogoff("greg.trasuk@webagesolutions.com");
                    sessionKey =
utilityWsProxy.customerLogin("greg.trasuk@webagesolutions.com",
"password");
                } else {
                    throw e;
                }
            }
            System.out.println("Received session key:" + sessionKey);
```

```
            /* Get anon customer id. */
            long customerId =
customerWsProxy.getCustomerByUsername("greg.trasuk@webagesolutions.com"
, sessionKey).getId();


 webCommsWsProxy.writeChatMessage(contactId, message, "",
CIChatMessageType.CHAT_MESSAGE_FROM_CUSTOMER, sessionKey);


function RequestCallBackNow($CallerName, $CallerNumber, $HomeDir)
{
  $firstName   = "ContactCenter";
  // similarly setup $lastname, $Username, $password, $number, $HomeDir
  require_once($HomeDir.'functions/Soap
Functions/reqister_new_customer.php');
  register(  $firstName,
  $lastName,
  $Username,
  $password,
  "",
  "",
  $number,
  $HomeDir);
  require_once($HomeDir.'functions/Soap Functions/login.php');
  $session_key = login($Username, $password, $HomeDir);
… continued…


function RequestCallBackNow($CallerName, $CallerNumber, $HomeDir)
{
  …continued from previous slide…
  require_once($HomeDir.'functions/Soap Functions/GetCustomerID.php');
  $cust_id = GetCustomerID($session_key , $Username, $HomeDir);
  require_once($HomeDir.'functions/Soap
Functions/get_default_outbound_skillset.php');
  $skillset_id = get_default_outbound_skillset($session_key, $HomeDir);

  require_once($HomeDir."functions/Soap
Functions/request_callback_now.php");
  request_call_now($session_key, $cust_id, $skillset_id, $CallerName,
"WebSite CallBack", "");

  return $Username;
}
$ajax->export("RequestCallBackNow", "page-
>RequestCallBackNow");//export function


function request_call_now($sessionkey, $cust_id, $skillset_id,
$details, $subject, $HomeDir)
{
```

```
  //require nesessary saop files
  require_once($HomeDir.'library/nusoap.php');

  //include config settings needed for this function
  include($HomeDir.'include/config.php');
  // define the soapaction as found in the wsdl
  $soapaction =
"http://webservices.ci.ccmm.applications.nortel.com/RequestImmediateCal
lback";

  // endpoint address
  $wsdl = $HTTP_REQUEST_TYPE.$CCMM_MACHINE_NAME.":".
$HTTP_RESQUEST_PORT."/ccmmwebservices/CICustomerWs.asmx";
  //create client object
  $client = new nusoap_client($wsdl, 'true');


(cont'd)
  //set decoding option
  $client->soap_defencoding = 'utf-8';
  //create soap message
  $mysoapmsg = $client->serializeEnvelope(
  '<?xml version="1.0" encoding="utf-8"?>
   …details ommitted
  </soap:Envelope>','',array(),'document', 'literal');

  // Send the SOAP message and specify the soapaction
  $response = $client->send($mysoapmsg, $soapaction);
  // Debugging output omitted…
  return;
}
```

```
4Login.cs
private void Login_Button_Click(object sender, EventArgs e)
    {
      //get user typed info
      username = Username_Field.Text;
      password = Password_Field.Text;
      Error_Message_Label.Text = "";
      //create link to our webservice
      OutboundUtilityWS.OutboundUtilityWS UtilInvoke = new
OutboundUtilityWS.OutboundUtilityWS();
      //invoke our webservice
      try
      {
        sessionkey = UtilInvoke.OutboundLogin(username, password);
      }
      catch(SoapException ex)
      {
        Error_Message_Label.Text = ex.Detail.InnerText;
        Error_Message_Label.Refresh();
        if(ex.Detail.InnerText.Contains("19021"))
        {
          //user is already logged in
          PromtLogout();
        }
      }
      if(sessionkey.Contains("__"))
      {
        //then this is a valid session key
        GlobalDef.getInstance().IsAccessGranted = true;
        GlobalDef.getInstance().MySessionKey = sessionkey;
        GlobalDef.getInstance().MyUserName = username;
        this.Hide();
        Main MainPage = new Main();
        MainPage.ShowDialog();
        Application.Exit();
      }
}
```

```
private void Refresh_Campaign_Button_Click(object sender, EventArgs e)
    {
      LoadCamapaigns();
    }
     private void LoadCamapaigns()
    {
      //create a link to the web service
      OutboundCampaignWS.OutboundCampaignWS campInvoke = new
OutboundCampaignWS.OutboundCampaignWS();

      //empty the array so that we dont re-add campaigns
      Campaign_Listbox.Items.Clear();
      //invoke the webservice
      MyActiveCampaigns =
campInvoke.GetCampaigns(GlobalDef.getInstance().MySessionKey);
      //add each active campaign name to the list
      if (MyActiveCampaigns.CampaignList != null)
      {
          if (MyActiveCampaigns.CampaignList.Length > 0)
          {
              for (int i = 0; i <
MyActiveCampaigns.CampaignList.Length; i++)
              {

Campaign_Listbox.Items.Add(MyActiveCampaigns.CampaignList[i].Name);
              }
          }
      }
}


protected override void Dispose(bool disposing)
    {
      // logoff agent
      Logoff();
      if(disposing && (components != null))
      {
        components.Dispose();
      }
      base.Dispose(disposing);
    }
    /// <summary>
    /// when the apps is exiitng, this is called to log the user off
    /// </summary>
    private void Logoff()
    {
      OutboundUtilityWS.OutboundUtilityWS UtilInvoke = new
OutboundUtilityWS.OutboundUtilityWS();
      //invoke our log out webservice
      try
      {
        UtilInvoke.OutboundLogoff(GlobalDef.getInstance().MyUserName);
```

```
        }
        catch(SoapException ex)
        {
          //error message here
        }
}
```