SIP Disaster Recovery

Genesys Interaction SDK land

1/5/2022

Table of Contents

SIP Disaster Recovery

Introduction

These pages provide guidelines for developers of agent desktop applications that are compatible with the Genesys SIP Business Continuity feature (also known as SIP Disaster Recovery), which is available in SIP Server 8.1. The pages address desktop applications based on the AIL client-side model. Code samples are included that illustrate the recommended design and coding techniques. Genesys recommends that you read these pages in conjunction with the *Framework 8.1 SIP Server High-Availability Deployment Guide,* as well as relevant AIL documentation.

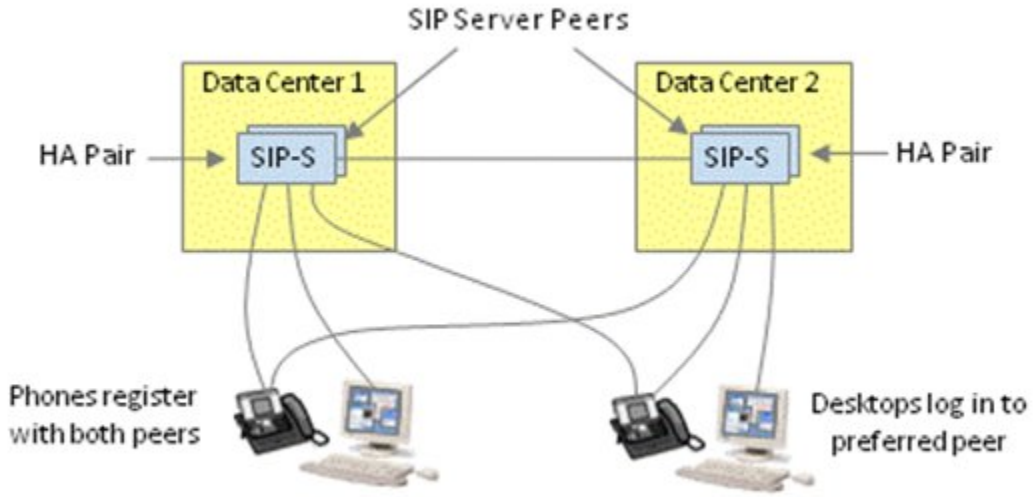
- SIP Business Continuity Architecture
- AIL-Based Desktops

SIP Business Continuity Architecture

The SIP Business Continuity Solution defines a formal architecture to support geographic redundancy (also known as Disaster Recovery) for deployments that include Genesys SIP Server 8.1 and higher. The solution supports pairs of active sites and provides site-wise redundancy, as well as local High Availability (HA) at each site.

SIP Server Peers

As shown in the following figure, SIP Business Continuity is based on a group of four SIP Servers configured as two HA pairs, which are known as SIP Server peers. The SIP Server peers collaborate to provide Active-Active redundancy between two sites, with full HA at each site. The solution can scale by adding additional groups of four SIP Servers.



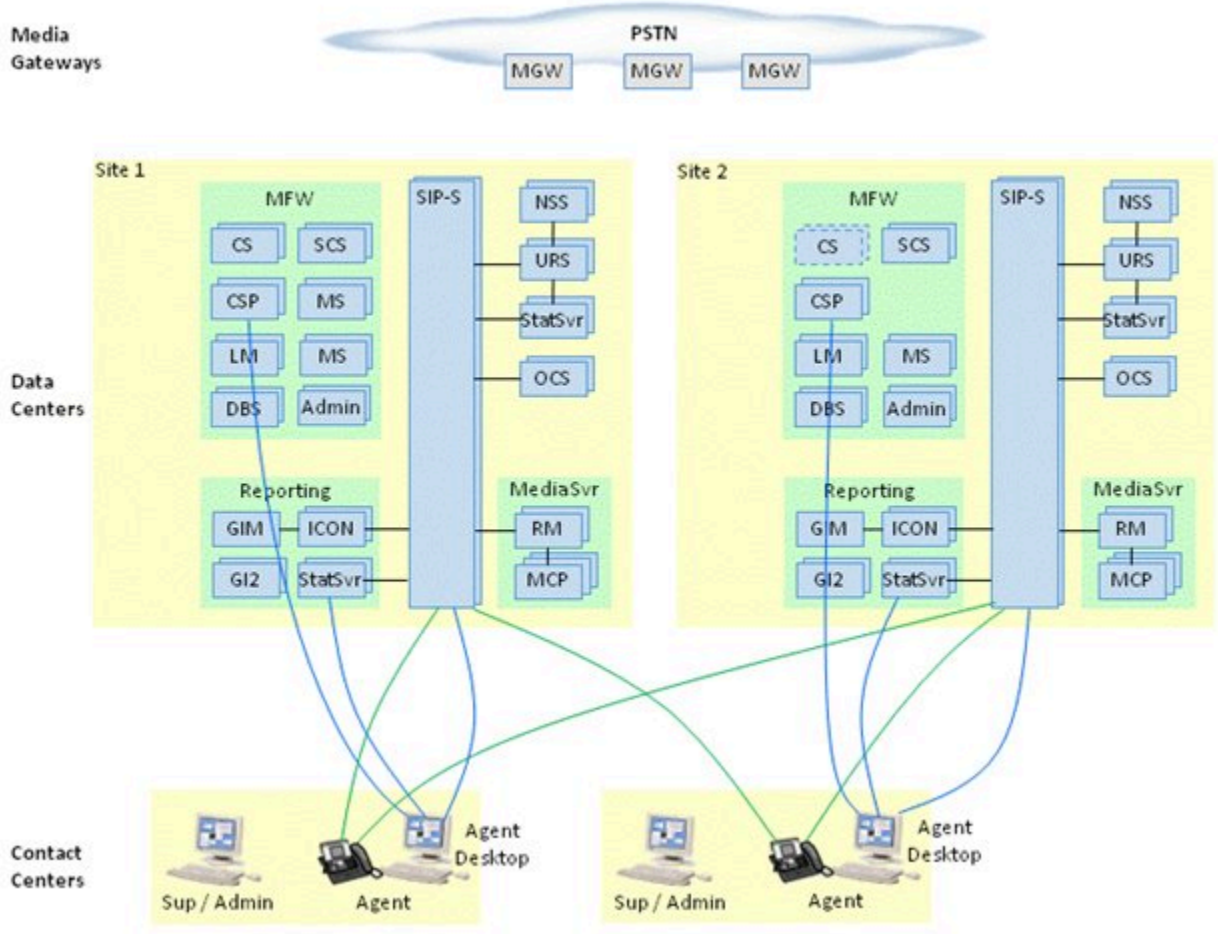
Agent SIP Phones are typically configured to maintain simultaneous registrations with both SIP Server peers, whereas Agent Desktop applications maintain a login to only one peer at any given time. The SIP Server peers collaborate to deliver calls to a given agent, including 3PCC calls and direct-dialed calls, via the peer at which the agent is currently logged in. Outbound 1PCC calls can be originated via either peer, but one peer is generally preferred based on phone configuration or DNS SRV priority. For more details about SIP Business Continuity, refer to the *Framework 8.1 SIP Server High-Availability Deployment Guide.*

Desktop Connections

The following figure illustrates the server applications to which the desktop application can connect. These are:

- Configuration Server Proxy. The desktop must connect and log in to Configuration Server Proxy to obtain configuration data for the desktop application and the agent that is logging in.
- SIP Server. As previously noted, the desktop must connect and log in to one of the SIP Server peers.
- Stat Server. The desktop application can optionally connect to Stat Server to obtain real-time report data.

These connections utilize the Genesys ConfigLib, Tlib, and StatLib protocols, respectively.



Each agent has a preferred site defined in the configuration. To provide the flexibility of configuration, the preferred site should be configurable hierarchically at the Tenant, Agent Group, and Person levels. When an agent logs in, the desktop, by default, connects and logins to the server application instances at the agent's preferred site. If the desktop is unable to log in initially to Configuration Server Proxy or SIP Server at the preferred site, or if it is subsequently unable to maintain its login to either server, it switches over its connection to the other site. These switchovers are referred to as Disaster Recovery (DR) switchovers.

Business Continuity Relationship to High Availability

All components of the SIP Business Continuity solution that support HA, including Configuration Server Proxy, SIP Server and Stat Server, are deployed with local HA at each site. In the event that the primary instance of any of these applications fails, Genesys Management Layer initiates an HA failover of that application to its backup instance. Genesys SDKs handle HA failovers through existing protocols; these HA failovers are transparent to desktop applications that utilize the SDKs. Additionally, Genesys applications that support Hot Standby High Availability (for example; SIP Server) employ synchronization (for example; call state, agent state), between the primary and backup members of an HA pair. This synchronization allows instantaneous recovery following a failure, with zero or minimal loss of stable, in-setup, or new calls. SIP Business Continuity does not involve the use of Genesys High Availability mechanisms or Genesys Management Layer and does not support synchronization between SIP Server peers. Genesys SDKs do not provide explicit support for SIP Business Continuity; therefore desktop applications must implement the Business Continuity logic discussed above at the application level. In the event of a site failure, the time typically taken for all components of the SIP Business Continuity solution to detect and recover from the failure is 60 seconds. During this time, it is expected that stable and in-setup calls, and some new calls, will be lost.

AIL-Based Desktops

This section describes the AIL-based desktops:

Configuration Data

The configuration data described in this section is not included as part of SIP Business Continuity. These configuration data items should be added to the customer's existing configuration. To ensure compatibility with future versions of SIP Business Continuity, third party desktops should adhere to the naming conventions described below.

Configuration Server

This section discusses desktop-related data that must be defined in the Genesys Configuration Server for use by the agent desktop application. **AIL Application Objects** Two ThirdPartyApplication objects must be configured: one for each site. Each AIL Application object must be configured with connections to the primary Configuration Server Proxy, SIP Server, and StatServer at the corresponding site. The SIP Server connections must be configured with an ADDP timeout of 20 to 30 seconds. Additionally, each AIL Application object must be configured with the following data, which is common to all agent desktops. The data must be placed into a section named agent-desktop of each application object.

Parameter	Description
site-timeout	Specifies the timeout interval, in seconds, after a connection loss to the High Availability (HA) Pair of SIP Servers, before the desktop should initiate a Disaster Recovery switchover. A typical value for this timeout is 30 seconds.
site-sip-register-timeout	Specifies the time interval, in seconds, to wait for SIP Phone registration before initiating a DR switchover, after the current SIP Phone connection was lost or its registration has expired. A typical value for this timeout is 30 seconds.

Person Configuration The person object for each agent must be configured with the following data to indicate the preferred site for the agent. The data must be placed into a section named agent-desktop of each person object.

Parameter	Description
site-1	Specifies the name of the AIL Application object corresponding to the site with the highest connection priority for the agent. This site is also referred to as the preferred site.
site-2	Specifies the name of the AIL Application object corresponding to the site with the second highest connection priority for the agent.

Desktop Local Configuration

This section describes data that must be available to the desktop to allow it to connect initially to Configuration Server Proxy in order to retrieve the configuration data described above. Depending on customer requirements, this data can be configured individually within each desktop instance, or can be configured in a local server from which all desktops obtain their initial configuration. In order for the desktop to be able to connect to either of the peer sites in the event that one site is down at the time of startup, the specified data items must be configured for both sites. At a minimum, the following local configuration must be available to the AIL desktop application:

- Primary Config Server Proxy IP address and port for both sites
- Backup Config Server Proxy IP address and port for both sites
- Application name and type for both sites
- Place name for both sites

Note: SIP Server Business Continuity requires that for a given agent phone with two DNs configured, a separate Place object be configured and associated with each DN.

Desktop Operation

Desktop Startup

On startup, the desktop application must first connect to the Configuration Server Proxy at one of the sites. Since the preferred site for the agent is not known at this point, this connection can be to either site. If the desktop cannot connect to the first site attempted, it should attempt to connect to the other site. To connect to Configuration Server Proxy, AIL requires the following parameters:

- IP address and port of Config Server Proxy - from local configuration
- AIL Application name - from local configuration
- Username and password - supplied by agent on logging into the desktop

The desktop connects to the Config Server Proxy using the Application name to identify itself, and the user name and password to identify and authenticate the agent. After connecting to Configuration Server Proxy, the desktop application should obtain the name of the preferred site (site 1) and secondary site (site 2) AIL Application Objects for the agent from configuration data for the agent's Person object. The desktop should then read the configuration for the preferred site AIL Application Object, in order to obtain the IP address and port number of the preferred Configuration Server Proxy. If the preferred Configuration Server Proxy for the agent is not the one to which the desktop is currently connected, the desktop should re-connect to the Configuration Server Proxy at the preferred site, according to the Disaster Recovery Switchover procedure described below. The desktop application may cache the preferred site information to avoid the possibility of a switchover immediately after startup; however, on connecting to Configuration Server Proxy, it should always follow the procedure described above, in case the configuration has changed. Caching should not be used if hotel seating / hot seating is employed. The associated techniques are illustrated in the code samples.

Detecting Site Failure

This section recommends the procedures that the desktop application should follow to detect a failure of the site to which it is currently connected. The desktop application should follow the same procedures regardless of whether it is currently connected to the preferred site or secondary site. The associated techniques are illustrated in the code samples. Genesys recommends that the desktop application create four listeners:

- A telephony service listener
- A configuration service listener
- A statistic service listener
- A listener to the DN of the agent's phone.

Telephony Service Listener

The telephony service represents the status of the connection between this application and the primary/backup pair of SIP Servers. If the telephony service goes to an OFF status, this indicates that both the primary and the backup SIP Server are unavailable. When this occurs, AIL tries to reconnect to the current primary/backup SIP Servers. The retry interval is configured in the constructor for AilLoader.
If the telephony service does not return to an ON status within the site-timeout interval, the desktop application should treat the condition as a site failure, and initiate a Disaster Recovery switchover according to the procedure described below.
If site-timeout is not configured, or has a value of zero, the desktop should not perform a Disaster Recovery switchover based on the status of the telephony service listener.

Configuration Service Listener

The configuration service represents the status of the connection between this application and the primary/backup pair of Configuration Servers. If the configuration service goes to an OFF status, this indicates that both the primary and the backup Configuration Servers are unavailable. When this occurs, AIL tries to reconnect to the current primary/backup Configuration Servers. The retry interval is configured in the constructor for AilLoader. Genesys recommends that the desktop application monitor the status of the configuration service. In the event that the service becomes unavailable, the desktop should warn the user and, if appropriate, generate alarms to upstream monitoring services, but should not initiate a Disaster Recovery switchover based solely on the status of this service.

Statistic Service Listener

The statistic service represents the status of the connection between this application and the primary/backup pair of Stat Servers. If the statistic service goes to an OFF status, this indicates that both the primary and the backup Stat Server are unavailable. When this occurs, AIL tries to reconnect to the current primary/backup Stat Servers. The retry interval is configured in the constructor for AilLoader.
Genesys recommends that the desktop application monitor the status of the statistic service. In the event that the service becomes unavailable, the desktop should warn the user and, if appropriate, generate alarms to upstream monitoring services, but should not initiate a Disaster Recovery switchover based solely on the status of this service.

DN Listener

The desktop application should utilize a DN Listener to detect `EventAgentLogout` events from SIP Server. There are several reasons why SIP Server can send `EventAgentLogout`:

- The agent logged out manually. The desktop application should take no further action.
- A supervisor forced a logout of the agent. The desktop application should take no further action. Genesys does not define a specific reason code for a supervisor forced logout, but recommends that, if possible, the customer's supervisor desktop assign a suitable reason code, and that the agent desktop check for that code.
- The associated SIP phone registration expired. In this case, the event data will contain the reason code `dr-force_logout`. If the desktop application does not have an embedded SIP endpoint, it should wait for the external endpoint to re-register. If the desktop application has an embedded SIP endpoint, it should wait for the endpoint to re-register or it should re-launch it if relevant. In either case, an `EventDNBackInService` event indicates successful re-registration.

If `site-sip-register-timeout` is configured with a non-zero value, and `EventDNBackInService` is not received within the time specified by site-sip-register-timeout, the desktop should initiate a Disaster Recovery switchover according to the procedure described below. If `site-sip-register-timeout` is not configured, or has a value of zero, the desktop should not perform a DR switchover based on the `dr-force_logout` reason code.

- SIP Server went into Graceful Shutdown mode. Graceful Shutdown mode is initiated through Genesys Administrator, and allows calls to be purged from SIP Server prior to shutting it down for maintenance

purposes. In this case, the event data will contain the reason code `graceful_shutdown_logout`. The desktop application should initiate a Disaster Recovery switchover according to the procedure described below.

Disaster Recovery Switchover Procedure

To effect a site switchover, it is necessary to restart AIL with the parameters for the target site. Genesys recommends that the killFactory() function be used. This allows AIL to be restarted without having to restart the underlying JVM. To minimize the startup time, set the option loading/on-demand to true. Once the connection to the other site has been established, the desktop application should set up Telephony Service and DN listeners for that site, as described above. The associated techniques are illustrated in the code samples.

Samples for AIL-Based Application

File bc.properties

```
PrimaryHost1 = bsgen803julien
PrimaryPort1 = 2020
BackupHost1 = bsgen803julien
BackupPort1 = 2030
ApplicationName1 = AIL_Client
PlaceName1 = Place_5320
UserName = SIP_5320
UserPass =
PrimaryHost2 = Suite80GAJulien
PrimaryPort2 = 2020
BackupHost2 = Suite80GAJulien
BackupPort2 = 2022
ApplicationName2 = AIL_Client
PlaceName2 = Place_5320
```

File BusinessContinuity.java

```
package com.genesyslab.ail.applicationblocks.businesscontinuity;
import java.io.*;
import java.util.*;
import com.genesyslab.ail.*;
import com.genesyslab.ail.event.*;
import org.apache.log4j.*;
public class BusinessContinuity {
  Logger mLogger;
  Properties mProperties;
  HashMap mSite1;
  HashMap mSite2;
  AilLoader mAL;
  AilFactory mAF;
  boolean mSwitchingOver;
  boolean mSwitchoverScheduled;
  int mSwitchoverDelay = 30;
  boolean mFirstConnection = true;
  ServiceListener mServiceListener;
  Place mPlace;
  Person mAgent;
  Dn mDN;
  DnListener mDnListener;
  int mCurrentSite = 1;
  Timer mTimer = new Timer();
  static BusinessContinuity mBC;
  public static void main(String[] s) {
      mBC = new BusinessContinuity();
  }
  public BusinessContinuity() {

      mLogger = org.apache.log4j.LogManager.getLoggerRepository().getLogger("BC");
```

```
       // This sample stores data in a property file as an example
       mProperties = new Properties();
       try {
           mProperties.load(new FileInputStream("bc.properties"));
       }
       catch (Exception e) {
           mLogger.error("Did not find bc.properties");
           System.exit(0);
       }
       // Connection parameters read in the file for site 1
       mSite1 = new HashMap();
       mSite1.put("PrimaryHost", mProperties.getProperty("PrimaryHost1"));
       mSite1.put("PrimaryPort", new Integer(mProperties.getProperty("PrimaryPort1")));
       mSite1.put("BackupHost", mProperties.getProperty("BackupHost1"));
       mSite1.put("BackupPort", new Integer(mProperties.getProperty("BackupPort1")));
       mSite1.put("ApplicationName", mProperties.getProperty("ApplicationName1"));
       mSite1.put("ApplicationType", AilLoader.ApplicationType.CLIENT);
       mSite1.put("UserName", mProperties.getProperty("UserName"));
       mSite1.put("UserPass", "");
       mSite1.put("PlaceName", mProperties.getProperty("PlaceName1"));
       // Connection parameters read in the file for site 2
       mSite2 = new HashMap();
       mSite2.put("PrimaryHost", mProperties.getProperty("PrimaryHost2"));
       mSite2.put("PrimaryPort", new Integer(mProperties.getProperty("PrimaryPort2")));
       mSite2.put("BackupHost", mProperties.getProperty("BackupHost2"));
       mSite2.put("BackupPort", new Integer(mProperties.getProperty("BackupPort2")));
       mSite2.put("ApplicationName", mProperties.getProperty("ApplicationName2"));
       mSite2.put("ApplicationType", AilLoader.ApplicationType.CLIENT);
       mSite2.put("UserName", mProperties.getProperty("UserName"));
       mSite2.put("UserPass", "");
       mSite2.put("PlaceName", mProperties.getProperty("PlaceName2"));
       // Initiate a logger for traces
       mLogger = org.apache.log4j.LogManager.getLoggerRepository().getLogger("BC");
       // Try and connect to the first site.
       mCurrentSite = 1;
       connect(mSite1);
   }
   private void connect(Map site) {
       try {
           mLogger.info("Connecting to " + site.get("PrimaryHost"));
           // Builds an AILLoader
           mAL = new AilLoader((String) site.get("PrimaryHost"), ((Integer)
site.get("PrimaryPort")).intValue(),
                               (String) site.get("BackupHost"), ((Integer)
site.get("BackupPort")).intValue(),
                               (String) site.get("UserName"), (String) site.get("UserPass"),
                               (String) site.get("ApplicationName"),
(AilLoader.ApplicationType) site.get("ApplicationType"),
                               15,  // This is the general  timeout for AIl requests
                               10); // This is the delay AIL waits when trying again to
connect to a server
           // Starts AIL
           mAF = mAL.getAilFactory();
           // At this point, mAF is null if the config server could not be reached for
instance
           if (mAF == null) {
               mLogger.error(mAL.getInitException());
               switchover();
           }
           // The serviceListener will listen and react on services status change
           mServiceListener = new ServiceListener() {
                   public void serviceStatusChanged(ServiceStatus.Type type, String name,
ServiceStatus.Status status) {
```

```
                    try {
                        if (status.toInt() == ServiceStatus.Status.OFF_) {
                            switch (type.toInt()) {
                            case ServiceStatus.Type.TELEPHONY_:
                            case ServiceStatus.Type.CONFIG_:
                            case ServiceStatus.Type.STAT_:
                                scheduleSwitchover();
                                break;
                            }
                        }
                    }
                    catch (Throwable t) {
                        mLogger.error("While processing serviceStatusChanged, ", t);
                    }
                }
            };
        // Listen to three types of services
        mAF.addServiceListener(ServiceStatus.Type.CONFIG, mServiceListener);
        mAF.addServiceListener(ServiceStatus.Type.STAT, mServiceListener);
        mAF.addServiceListener(ServiceStatus.Type.TELEPHONY, mServiceListener);
        // In this sample, the UserName UserPassword and PlaceName are stores in the
properties file.
        // You probably want to read them from a login banner.
        mPlace = mAF.getPlace((String) mProperties.getProperty("PlaceName" +
mCurrentSite));
        mLogger.info("Place is " + mPlace);
        mAgent = mAF.getPerson((String) mProperties.getProperty("UserName"));
        Map annex = mAgent.getAnnex();
        try {
            Map section = (Map) annex.get("agent-desktop");
            if (mFirstConnection) {
                String preferred = (String) section.get("disaster-recovery.preferred-
site");
                if (!mProperties.getProperty("ApplicationName1").equals(preferred)) {
                    // Switch Site 1 and site 2 in the properties file or wherever you
store that info.
                }
                mSwitchoverDelay = Integer.parseInt((String) section.get("disaster-
recovery.timeout"));
                mFirstConnection = false;
            }
        }
        catch (Exception e) {
            mLogger.warn("Agent "  + mAgent.getName() + " does not have the expected
annex");
        }
        mDN = (Dn) mPlace.getDns().iterator().next();
        // Listens to the DN status and initiates a switchover on graceful shutdown.
        mDnListener = new DnListener() {
                public void handleDnEvent(DnEvent event) {
                    try {
                        if (event.getStatus().toInt() == Dn.Status.LOGGED_OUT_ ||
event.getStatus().toInt() == Dn.Status.BUSY_LOGGED_OUT_) {
                            Map tExtensions = event.getTEventExtensions();
                            if (tExtensions != null) {
                                String reasonCode = (String) tExtensions.get("ReasonCode");
                                if ("graceful_shutdown_logout".equals(reasonCode)) {
                                    switchover();
                                }
                            }
                        }
                    } catch (Throwable t) { mLogger.error("While processing DNEvent, ",
t); }
```

```
                    }
                    public void deleted() { }
                    public void contactChanged(InteractionEvent ie) { }
                    public void handleInteractionEvent(InteractionEvent ie) { }
                };
            if (mDN != null) {
                mDN.addDnListener(mDnListener);
            }

            mLogger.info("DN is " + mDN);
        } catch (Throwable t2) { mLogger.error("While connecting, ", t2); }
    }
    // Before actually doing a switchover, we want to give a chance to the server/network to
get back in service
    private void scheduleSwitchover() {
        if (!mSwitchoverScheduled) {
            mSwitchoverScheduled = true;
            mLogger.info("Scheduling switchover");
            // Schedule the switchover in a timer task. We don't want to wait in this thread.
            // We are in a callback thread because this method is called from listeners.
            mTimer.schedule(new TimerTask() {
                    public void run() {
                        Map services = mAF.getServices();

                        if (getService(services, ServiceStatus.Type.TELEPHONY).mStatus.toInt()
== ServiceStatus.Status.ON_
                            && getService(services, ServiceStatus.Type.CONFIG).mStatus.toInt()
== ServiceStatus.Status.ON_
                            && getService(services, ServiceStatus.Type.STAT).mStatus.toInt()
== ServiceStatus.Status.ON_) {
                            mLogger.info("Services are ON. Switchover cancelled");
                            // Don't switchover if the services are all ON.
                            mSwitchoverScheduled = false;
                            return;
                        }
                        else {
                            switchover();
                            mSwitchoverScheduled = false;
                        }
                    }
                },
                mSwitchoverDelay * 1000);
        }
    }
    private ServiceStatus getService(Map services, ServiceStatus.Type type) {
        for (Iterator it = services.values().iterator() ; it.hasNext() ; ) {
            ServiceStatus ss = (ServiceStatus) it.next();
            if (ss.mType.toInt() == type.toInt()) {
                return ss;
            }
        }
        return null;
    }
    // The switchover procedure itself
    // It invokes the timer in order to release the thread that calls
    // because it is probably a callback thread and we don't want to
    //
    private void switchover() {
        try {
            if (mSwitchingOver) {
                mLogger.warn("Already switching over");
                return;
            }
```

```
            mSwitchingOver = true;
            mLogger.info("Switching over. Current site was " + mCurrentSite);
            if (mDN != null) { mDN.removeDnListener(mDnListener); }
            if (mAF != null) { mAF.removeServiceListener(ServiceStatus.Type.TELEPHONY,
mServiceListener); }
            // Release all the references to the objects coming from the previous factory
            mServiceListener = null;
            mDnListener = null;
            mPlace = null;
            mDN = null;
            mAF = null;
            // Kill the old AilFactory
            mAL.killFactory();
            if (mCurrentSite == 1) {
                mCurrentSite = 2;
                connect(mSite2);
            }
            else {
                mCurrentSite = 1;
                connect(mSite1);
            }
            mLogger.info("Switchover completed. Current site is now " + mCurrentSite);
            mSwitchingOver = false;
        }
        catch (Throwable t) {
            mLogger.error("While running switchover, ", t);
        }
    }
}
```