Events

This chapter provides an overview of the events that the RDU and DPEs provide, and explains how to register and handle these events.

Overview

Using the Prime Cable Provisioning RDU Java client library, you can register for numerous types of events, which are sourced from the RDU and the DPEs. The events that are sourced include:

- Device notification.
- Asynchronous operation notification.
- Batch status events.
- Custom extension events.
- Policy related events.

Event Registration

Events are registered by implementing the appropriate event listener interface. The resulting class is then registered from the PACE connection along with a qualifier.

The qualifier further filters the events that the client receives. If the client wants to receive all events, you can use the QualifyAll qualifier. For an object that can be modified in the RDU, a corresponding event is available in the API. For a complete list of available events, see the *package.com.cisco.provisioning.cpe.events* section in the API Javadocs.

Each event class has a specific qualifier with methods that allow you to refine the events that are to be delivered to the registered listener.

Note You can use only the qualifiers that the RDU Java client library provides. Prime Cable Provisioning does not support implementing your own qualifiers.

For example, to handle all asynchronous operation events that are fired when an on-connect device operation completes:

Step 1 Create the listener class using:

```
public class AsyncEventHandler implements AsyncOperationListener
{
    private boolean m_isOneShot;

    /**
     * The method invoked when a {@link AsyncOperationEvent
     * AsyncOperationEvent} arrives as a result of an async
     * operation completing.
     *
     * <P>
     * @param ev The object containing the {@link AsyncOperationEvent
     * AsyncOperationEvent} data.
     */
    public void completed(final AsyncOperationEvent ev)
    {
        // handle the incoming event
    }

    /**
     * Gets oneShot mode value, specifying whether or not the listener
     * is registered for just one occurrence of the Event.
     *
     * <P>
     * @return <TT>true</TT> if oneShot mode has been set.
     */
    public boolean getOneShot()
    {
        return m_isOneShot;
    }

    /**
     * Sets oneShot mode, specifying that the registration request is
     * for just one occurrence of the Event.
     *
     * <P>
     * @param flg <TT>true</TT> if oneShot mode is being set.
     */
    public void setOneShot(final boolean flg)
    {
        m_isOneShot = flg;
    }
}
```

Step 2 Register the created listener class using the PACE connection:

```
final AsyncEventHandler handler = new AsyncEventHandler();
// use a qualifier that filters all events
final Qualifier qualifier = new QualifyAll();

// register the listener, this contact the RDU
// and from now on we start receiving events
connection.addAsyncOperationLister(handler, qualifier);
```

Note If the connection breaks after the listener is registered, you do not have to reregister the listener. The RDU Java client library automatically registers the listener again.

Step 3 Receive the events. The listener class can be executed when the event arrives.

Step 4 Remove the listener that is created.

You can use any of the following methods to remove a listener:

- Where the implementing class can specify if the listener is one shot. This means that the listener receive only the first qualified event and is removed after receiving its first event.
- By using the PACE connection with an explicit remove listener call.

To explicitly remove the event listener that was created in Step 1:

```
// unregister the listener
// note we must use the same references for the handler
// and the qualifier from the addAsyncOperationListener
// method call
connection.removeAsyncOperationLister(handler, qualifier);
```

Event Handling

When an event is delivered to your registered listener, you must execute any logic that is required. However, because the thread delivering the event does so from the Prime Cable Provisioning RDU Java client library, you must exercise caution.

When running any logic for handling events:

- Avoid any complex logic for your registered listener that uses a Prime Cable Provisioning RDU Java client library thread. If the thread is busy processing the listener, the thread may not be able to deliver events to other listeners or batch results to threads that have completed synchronous posting.
- Re-accessing the PACE connection can cause a deadlock. For example, if you receive an event and then try to submit a new batch while handling the event with the current thread, a deadlock can occur in the RDU Java client library.

To avoid these issues, we recommend that you:

- Keep the logic in your listener short.
- Avoid re-accessing the PACE connection. If you require a more complex logic, you can notify any one of your threads for the processing.

Event Reliability

The RDU Java client library receives events when it maintains a connection with the RDU. If the connection is lost (for example, because of a network crash), events may be lost. You cannot retrieve missed events.

You may also lose events that are generated from the DPE. For example, an interruption in the connection from the DPE to the RDU makes it impossible for the DPE to forward the events to the RDU, and from there, to the client.

For more information on how the RDU Java client library communicates with the RDU, see:

- Chapter 6, “Getting Started with the Prime Cable Provisioning API.”
- Chapter 7, “Java API Client Use Cases.”