Cisco Physical Access Manager API Reference Guide

Release 1.5.0

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Text Part Number: OL-31223-01
CONTENTS

Preface ix
Audience ix
Organization ix
Obtaining Documentation, Obtaining Support, and Security Guidelines x

CHAPTER 1

Overview 1-1
Contents 1-1
Supported Functionality 1-2
Understanding Cisco PAM Web Services 1-2
   Interfaces 1-3
   Bindings and Endpoint Address 1-4
   Viewing the Web Service Information on a Cisco PAM server 1-4
   Security 1-4
   Error Handling 1-5
   Request and Response Samples 1-5
   Auto-Generating Java or C# Code from the WSDL File 1-5
      Client Application Code—Java Example 1-5
      Client Application Code—C# Example 1-6
   Namespaces 1-8
Enabling Web Services on the Cisco PAM Server 1-8
   Enabling the API Service on the Cisco PAM Server 1-8
   Purchasing and Installing the Cisco PAM API License 1-9
      Purchasing the API License 1-10
      Installing the API License 1-10
      Verifying the Installed Licenses 1-12
      Displaying the Cisco PAM Appliance Serial Number 1-12
Authentication and Authorization 1-13
API Security 1-14
Understanding Unique IDs 1-14
API Logging 1-15
CHAPTER 2

API Functions 2-1

Function Summary 2-1

API Functions 2-6

Authentication APIs 2-7

authenticateUser 2-7

logoutUser 2-7

Physical Security Integration Management (PSIM) APIs 2-8

getAcDeviceTypes 2-8

gEventTypes 2-8

getAllEventsCategories 2-9

getAllCameraDevices 2-10

gAllAcDevices 2-12

gDeviceChildren 2-13

setDeviceAdminState 2-14

getAllDoorGroups 2-15

getAllDoorsByGroup 2-15

gAllPerson 2-16

gAllPersonTypeIds 2-17

gAllPersonStatusIds 2-17

gAllPersonByType 2-18

enrollPerson 2-18

updatePerson 2-19

gPersonInfoByPersonId 2-20

gCustomFieldsByPersonnel 2-21

gAllOrganizations 2-21

gAllDepartments 2-22

gAllEventsByCategory 2-22

gAllEventsByType 2-23

gUserAccessEvents 2-25

gAccessEventsByDoor 2-26

gAllAlarmsByCategory 2-28

gAllAlarmsByType 2-29

gAllAlarmUpdates 2-30

gAlarmInstructions 2-31

clearAlarm 2-32

ackAlarm 2-34

gVersionInfo 2-34

gAlarmDuplicates 2-35

Event Notification 2-36
registerAcEventCallback  2-36
registerAcEventXMLCallback  2-37
unregisterAcEventCallback  2-38
notifyAcEvent  2-39
Example to Implement and Register a Service Interface  2-39
Door Command APIs  2-41
grantDoorAccess  2-41
setDoorMode  2-41
resetDoorMode  2-41
getDoorMode  2-42
setDoorModeConfig  2-42
applyScheduleToDoor  2-43
getAssociatedGWDoor  2-44
Badge Enrollment APIs  2-44
getAllAccessLevels  2-44
getAllAccessLevelNames  2-45
getAccessLevelByName  2-45
getAllScheduleNames  2-46
getScheduleByName  2-47
enrollBadge  2-47
assignAccessLevels  2-49
getPersonInfoByBadge  2-49
getAllBadgesByPerson  2-50
getBadgeValidityTypes  2-51
getCredentialTemplateName  2-51
getBadgeRoles  2-52
getAllBadgeTemplateName  2-52
getAccessLevelByPerson  2-53
getBadgeByCardNum  2-53
updateBadge  2-54
getAllUpdatedBadges  2-56
getAllUpdatedPersonnels  2-56
getAccessLevelByUnid  2-57
getAllBadgeTypeIds  2-58
getAllBadgeRefs  2-58
APIs for Time and Schedule based on Location  2-60
createTimeEntry  2-60
updateTimeEntry  2-60
deleteTimeEntry  2-61
getAllTimeEntries  2-61
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTimeEntryById</td>
<td>2-63</td>
</tr>
<tr>
<td>getTimeEntryByName</td>
<td>2-63</td>
</tr>
<tr>
<td>getTimeAllSchedulesForTimeEntry</td>
<td>2-65</td>
</tr>
<tr>
<td>createTimeRangeGroup</td>
<td>2-65</td>
</tr>
<tr>
<td>updateTimeRangeGroup</td>
<td>2-67</td>
</tr>
<tr>
<td>deleteTimeRangeGroup</td>
<td>2-67</td>
</tr>
<tr>
<td>getTimeRangeGroupById</td>
<td>2-69</td>
</tr>
<tr>
<td>getTimeRangeGroupByName</td>
<td>2-69</td>
</tr>
<tr>
<td>createTimeEntryCollection</td>
<td>2-69</td>
</tr>
<tr>
<td>updateTimeEntryCollection</td>
<td>2-71</td>
</tr>
<tr>
<td>deleteTimeEntryCollection</td>
<td>2-71</td>
</tr>
<tr>
<td>getTimeAllEntryCollections</td>
<td>2-72</td>
</tr>
<tr>
<td>getTimeEntryCollectionById</td>
<td>2-72</td>
</tr>
<tr>
<td>getTimeEntryCollectionByName</td>
<td>2-73</td>
</tr>
<tr>
<td>getTimeAllEntryCollections</td>
<td>2-73</td>
</tr>
<tr>
<td>createTimeEntryCollectionForTimeEntryGroup</td>
<td>2-73</td>
</tr>
<tr>
<td>createSchedule</td>
<td>2-75</td>
</tr>
<tr>
<td>updateSchedule</td>
<td>2-75</td>
</tr>
<tr>
<td>addExceptionToSchedule</td>
<td>2-77</td>
</tr>
<tr>
<td>deleteSchedule</td>
<td>2-77</td>
</tr>
<tr>
<td>getTimeAllSchedules</td>
<td>2-78</td>
</tr>
<tr>
<td>getAllDoorSchedules</td>
<td>2-78</td>
</tr>
<tr>
<td>getTimeAllSchedulesForTimeRangeGroup</td>
<td>2-80</td>
</tr>
<tr>
<td>APIs for Location Listing</td>
<td>2-81</td>
</tr>
<tr>
<td>getAllLocationsInScope</td>
<td>2-81</td>
</tr>
<tr>
<td>getAncestorLocations</td>
<td>2-81</td>
</tr>
<tr>
<td>API for Configuration changes</td>
<td>2-83</td>
</tr>
<tr>
<td>applyConfigurationChanges</td>
<td>2-83</td>
</tr>
<tr>
<td>applyConfigurationChangesToLocation</td>
<td>2-83</td>
</tr>
<tr>
<td>Additional Information</td>
<td>2-83</td>
</tr>
<tr>
<td>Precursor to APIs based on Location</td>
<td>2-83</td>
</tr>
<tr>
<td>Sample Usecases</td>
<td>2-84</td>
</tr>
<tr>
<td>Recording External Events</td>
<td>2-85</td>
</tr>
<tr>
<td>Define External Event Types Using Event Format</td>
<td>2-86</td>
</tr>
<tr>
<td>Create a Text File to Define Event Names PAM</td>
<td>2-86</td>
</tr>
<tr>
<td>Import the Files into PAM</td>
<td>2-87</td>
</tr>
<tr>
<td>recordExtEvent</td>
<td>2-89</td>
</tr>
<tr>
<td>Examples: Using Java and C# Code</td>
<td>2-89</td>
</tr>
</tbody>
</table>
CHAPTER 3  Fault Codes  3-1

INDEX
Preface

This document provides the information that is required to understand and use the Cisco Physical Access Control application programming interface (API).

Audience

This document is intended for developers who want to use the Cisco Physical Access Control API to control various access control features and functions. It assumes that developers have knowledge or experience with Cisco Physical Access Control, a high-level programming language, and the following:

- Extensible Markup Language (XML)
- XML Schema
- Web Services
- Hyper Text Transport Protocol (HTTP) or Secure HTTP (HTTPS)
- Simple Object Access Protocol (SOAP)

Organization

This document is organized as follows:

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, “Overview”</td>
<td>Introduces the API, describes the WSDL and API URLs, authentication method, security and API logging.</td>
</tr>
<tr>
<td>2, “API Functions”</td>
<td>Provides a summary of each API function and describes each function in detail.</td>
</tr>
<tr>
<td>3, “Fault Codes”</td>
<td>Provides fault codes and descriptions for API errors.</td>
</tr>
</tbody>
</table>
Obtaining Documentation, Obtaining Support, and Security Guidelines

For information about obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and recommended aliases and general Cisco documents, see the monthly What's New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:

Overview

Access Control Web Services (ACWS) defines an application programming interface (API) for the Cisco Physical Access Manager (Cisco PAM).

This chapter includes general information, and instructions to enable the ACWS API on a Cisco PAM server. It also describes the ACWS authentication method, and the Namespaces and other information used to issue API requests.

Contents

- Supported Functionality, page 1-2
- Understanding Cisco PAM Web Services, page 1-2
  - Interfaces, page 1-3
  - Bindings and Endpoint Address, page 1-4
  - Viewing the Web Service Information on a Cisco PAM server, page 1-4
  - Security, page 1-4
  - Error Handling, page 1-5
  - Request and Response Samples, page 1-5
  - Auto-Generating Java or C# Code from the WSDL File, page 1-5
  - Namespaces, page 1-8
- Enabling Web Services on the Cisco PAM Server, page 1-8
  - Enabling the API Service on the Cisco PAM Server, page 1-8
  - Purchasing and Installing the Cisco PAM API License, page 1-9
- Authentication and Authorization, page 1-13
- API Security, page 1-14
- Understanding Unique IDs, page 1-14
- API Logging, page 1-15
Supported Functionality

The API for Cisco Physical Access Control Release 1.4.1 and higher supports the following features:

- Authentication APIs—Use these APIs to retrieve a security context object before calling any other API. The object is provided as a parameter in all subsequent calls for that API session. If the session ends, a new object must be retrieved.

- Physical Security Integration Management (PSIM)—Use these APIs in Physical Security Operations Management applications. These APIs return information on access control devices, users, events and alarms. The API provides mechanisms to query events or alarms based on event type, time-interval, and source device criteria.

- Event Notification—Use these APIs to notify a client application when an event or alarm occurs. You can also query events or alarms based on the event type, time-interval, or source device.

- Door Command APIs—Use these APIs to trigger actions based on access control events. For example, when a user attempts to access a door or device, these APIs can open or close the door.

- Badge Enrollment APIs—Use these APIs to provision badge credentials in the access control system. Also returns information on access levels and schedules.

- Location-based Time and Schedule APIs—Use these APIs to trigger action based on time and schedule entries.

- Location Listing APIs—Use these APIs to list the hierarchy of the logged in user’s location.

- API for Configuration changes—Use this API to push the changes to gateways in the logged in user's location scope.

- Record External Events—Use this API to record an external event or alarm in Cisco PAM.

Understanding Cisco PAM Web Services

The Cisco PAM web service (PSIMWsService) can provide information regarding doors, locks, badges, personnel, and alarms. The APIs can also execute door commands, act on an alarm, or update badges. Following are the examples of PSIMWsService:

- An application can receive the events generated when user access is granted or denied.
- The application can open or close a door.
- An application can create visitors and assign access policies to allow access to specific doors or locations.
- An application can provision badge credentials in the access control system.

This section describes the following:

- Interfaces, page 1-3
- Bindings and Endpoint Address, page 1-4
- Viewing the Web Service Information on a Cisco PAM server, page 1-4
- Security, page 1-4
- Error Handling, page 1-5
- Request and Response Samples, page 1-5
Interfaces

The Cisco PAM web service (PSIMWsService) exposes two interfaces: PSIMWsPortType and AccessPolicyPortType (Figure 1-1).

Tip

The interface is also known as portType in web service terminology. The interfaces are exposed using WSDL 1.1 specifications.

The PSIMWsPortType interface provides the following methods:
- Authenticate users (login and logout)
- Get details of access control device, including doors and locks
- Get personnel and organization details
- Act on an alarm
- Execute a door command
- Register a listener to receive Cisco PAM events

The AccessPolicyPortType interface provides the following methods:
- Get details of access levels and schedules
- Get details of badges
- Update badges

Figure 1-1 CPAM Web Service Interfaces

Note

The PSIMWsService web service does not provide methods to configure the Cisco PAM server. Use the Cisco PAM client to configure the Cisco PAM server.
Bindings and Endpoint Address

Since both interfaces support SOAP/HTTP or XML/HTTP message formats (Figure 1-1), there are a total of four unique endpoint addresses (or URIs). Each endpoint address uniquely identifies the interface and the message format to use.

The client application uses one of the endpoint addresses shown in Table 1-1 to access the web-service method, based on the message format (SOAP or XML).

Table 1-1 Interface Message Formats and Endpoint Addresses

<table>
<thead>
<tr>
<th>Interface</th>
<th>Message Format</th>
<th>Endpoint Address used by client application</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSIMWsPortType</td>
<td>SOAP</td>
<td>http://&lt;cpam-server&gt;/acws/services/psimws</td>
</tr>
<tr>
<td>PSIMWsPortType</td>
<td>XML</td>
<td>http://&lt;cpam-server&gt;/acws/services/psimxml</td>
</tr>
<tr>
<td>AccessPolicyPortType</td>
<td>SOAP</td>
<td>http://&lt;cpam-server&gt;/acws/services/acpolicy</td>
</tr>
<tr>
<td>AccessPolicyPortType</td>
<td>XML</td>
<td>http://&lt;cpam-server&gt;/acws/services/acpolicyxml</td>
</tr>
</tbody>
</table>

Note: The implementation for many individual methods using XML/HTTP format is not complete. Please refer to the actual WSDL or send an e-mail to acws-interest@cisco.com for more information.

Viewing the Web Service Information on a Cisco PAM server

Enter the following URL to display the interfaces, methods and endpoint addresses available on a Cisco PAM server.

http://<cpam-server>/acws/services

Note: The ACVSMPortType interface methods are not supported. These methods provide integration between Cisco PAM and Cisco Video Surveillance Manager (Cisco VSM) and should not be used by third party applications.

Enter the following URLs to display the actual WSDL for each supported interface.

http://<cpam-server>/acws/services/psimws?wsdl
http://<cpam-server>/acws/services/acpolicy?wsdl

Security

The Cisco PAM Web Service provides the following security measures:

- HTTP or HTTPS based access.
- Application level authentication. Before calling any method, the client application must call authenticateUser. This method returns a security context object used as a required parameter in subsequent API calls.
Chapter 1      Overview

Error Handling

Method errors or exceptions return an AcWsFault that include a description that identify if the event as major or minor fault code.

See Chapter 3, “Fault Codes” for descriptions of the major fault codes.

Request and Response Samples

Request and response examples are provided in Chapter 2, “API Functions”. To capture additional examples, use a network packet capture tool such as the TCPMon utility.

Auto-Generating Java or C# Code from the WSDL File

This section describes tools used to auto-convert a WSDL document to fully annotated Java or C# code that can be used to call the Cisco PAM web service methods.

- Client Application Code—Java Example, page 1-5
- Client Application Code—C# Example, page 1-6

Client Application Code—Java Example

Use the wsdl2java converter tool to convert a WSDL document to fully annotated Java code that can be used to access the Cisco PAM Web Service methods.

Step 1
Enter the wsdl2java command to convert a WSDL document to fully annotated Java code. The syntax is:

```
wsdl2java -d <destination-folder> <wsdl-url>
```

For example, the following command generates the Java code in the /generated-source folder:

```
wsdl2java -d /generated-source http://my-cpam-server/acws/services/psimws?wsdl
```

Tip
See https://cwiki.apache.org/CXF20DOC/wsdl-to-java.html for more information on wsdl2java converter tool.

Step 2
Compile the code and use it in your client application to integrate with CPAM WS.

Step 3
Call the Web Service methods.

For example:

```
// The PSIMWsService Web Service
private final static QName SERVICE = new QName("http://cisco.com/physec/acws/", "PSIMWsService");
// WSDL location URL
private final static URL WSDL_LOCATION = new URL("http://cpam-server/acws/services/psimws?wsdl");

PSIMWsService _webService = null;
PSIMWsPortType m_PSIMWsInterface = null;
AccessPolicyPortType m_AccessPolicyInterface
```
SecurityContext _ctx = null;

...

try {
    //Reference to the PSIMWsService web service
    _webService = new PSIMWsService(WSDL_LOCATION, SERVICE);
    //Get handle to PSIMWsPortType interface
    _PSIMWsInterface = _webService.getPSIMWsSoapPort();
    //Get handle to AccessPolicyPortType interface
    _AccessPolicyInterface = _webService.getAccessPolicySoapPort();
} 

catch (WebServiceException e) {
    //Handle exception
}

catch (Exception e) {
    //Handle exception
}

...

// Authenticate user
UserCredentialType uct = new UserCredentialType();
uct.setUsername("username");
uct.setPassword("password");
_ctx = _PSIMWsInterface.authenticateUser(uct);

...

//Call methods using interface handles
AcDeviceTypeList deviceTypeList = _PSIMWsInterface.getAcDeviceTypes(_ctx);
...
List<Schedule> schedules = _AccessPolicyInterface.getAllSchedules(_ctx);
...

//In the end logout user and close the session
_PSIMWsInterface.logoutUser(_ctx);

---

Client Application Code—C# Example

Use the \texttt{wsdl.exe} converter tool to convert a WSDL document to fully annotated C# code that can be used to access the Cisco PAM Web Service methods.

\textbf{Step 1} Enter the \texttt{wsdl.exe} command to convert a WSDL document to fully annotated C# code.

The syntax is:

```
wsdl /l:CS /protocol:SOAP <wsdl-file> <xsd files ...>
```

The following example generates the PSIMWsService.cs C# code:

```
wsdl /l:CS /protocol:SOAP acws.wsdl physic-common.xsd ws-addr.xsd ac-common.xsd
ac-schedule.xsd ac-policy.xsd ac-event.xsd
```
Tip

- If you install MS Dev Studio 2008 and .NET Framework SDK 3.5 then you already have the WSDL converter tool.
- See http://msdn.microsoft.com/en-us/library/7h3ystb6(VS.71).aspx for more information on the \wsdl.exe\ tool.

Step 2

Create a Client Proxy DLL using the C# command line compiler.

The following example generates the PSIMWsService.dll. Add reference to this DLL in the MS Dev Studio for your C# application.


Tip


Step 3

Call the Web Service methods.

For example:

```csharp
// The PSIMWsService Web Service
private String _PSIMServiceUrl = "http://cpam-server/acws/service/psimws";
private String _AccessPolicyServiceUrl = "http://cpam-server/acws/service/acpolicy";
private PSIMWsBinding _psimBinding;
private AccessPolicyBinding _apBinding;
private SecurityContext _ctx;
...

// PSIMWsPortType interface with SOAP/HTTP
_ psimBinding = new PSIMWsBinding();
_ psimBinding.Url = _PSIMServiceUrl;

// AccessPolicyPortType interface with SOAP/HTTP
_ apBinding = new AccessPolicyBinding();
_ apBinding.URL = _AccessPolicyServiceUrl;
...

// Authenticate user
UserCredentialType uct = new UserCredentialType();
uct.username = "username";
uct.password = "password";
_ ctx = _ psimBinding.authenticateUser(uct);
...

// Call methods using interface handles
getAcDeviceTypesReqType reqType = new getAcDeviceTypesReqType();
reqType.secCtx = _ctx;
getAcDeviceTypesResponse resp = _ psimBinding.getAcDeviceTypes(reqType);
AcDeviceType[] dtypes = resp.deviceTypes;
...

getAllAccessLevels gaal = new getAllAccessLeves();
gaal.ctx = _ctx;
accessLevel[] als = _apBinding.getAllAccessLevels(gaal);
```
---

```java
// In the end logout user and close the session
logoutUser lu = new logoutUser();
lu.setCtx = _ctx;
_psimBinding.logoutUser(lu);
```

## Namespaces

Web Services APIs are defined using WSDL and various object types defined by the XML schema. The schema definitions uses the following target namespaces:

- Interfaces, methods and types are defined using the target namespace:
  http://cisco.com/physec/acws
- BaseDevice is defined using the target namespace:
- Camera devices are defined using the target namespace:
- ACWS devices and video names are derived by extension.

### Enabling Web Services on the Cisco PAM Server

To enable the Web Services API functionality on the Cisco PAM server, you must purchase and install the optional Web Services license, and enable the API service, as described in the following sections:

- Enabling the API Service on the Cisco PAM Server, page 1-8
- Purchasing and Installing the Cisco PAM API License, page 1-9

### Enabling the API Service on the Cisco PAM Server

In Cisco PAM Release 1.2.0, you must manually enable the Web Service API, as described in the following procedure.

Beginning with Cisco PAM Release 1.3.0, the Web Service API is enabled by default. Use the following instructions to verify the Status is Enabled.

For all releases, you must install the API license, as described in the “Purchasing and Installing the Cisco PAM API License” section on page 1-9.

#### Procedure

Complete the following procedure to verify that the Web Services API is enabled, or to enable the service if it is stopped.

**Step 1** Log on to the Cisco PAM appliance as described in the *Cisco Physical Access Manager User Guide*. 
Step 2
Click the Monitoring tab and then click Status, as shown in Figure 1-2.
The Status page appears by default. This page also appears when you first log on.

Figure 1-2 Services tab in the Cisco PAM Server Administration Utility

Step 3
Verify that the status is Enabled.

Step 4
If the status is Disabled, click the Enable button.
A confirmation message appears and the Status changes to Enabled.

Step 5
Continue to the “Purchasing and Installing the Cisco PAM API License” section on page 1-9.

Note
Beginning in Release 1.3.0, the message Web Services license not applied appears if the API license is not installed.

Purchasing and Installing the Cisco PAM API License

To enable the API functionality, you must purchase the optional API license from the Cisco website and install it on the Cisco PAM server. If the API license is not installed, API requests to the Cisco PAM server return an error.

This section includes the following information:
- Purchasing the API License, page 1-10
- Installing the API License, page 1-10
- Verifying the Installed Licenses, page 1-12
- Displaying the Cisco PAM Appliance Serial Number, page 1-12

Tip
For more information on server configuration and optional licenses, see the Cisco Physical Access Manager User Guide.
Purchasing the API License

To purchase the Cisco PAM API license, do the following:

**Step 1** Determine the Cisco PAM appliance serial number (the serial number is required to complete the purchase). See Displaying the Cisco PAM Appliance Serial Number, page 1-12 for more information.

**Step 2** Purchase the licence by contacting your Cisco sales representative or any Cisco reseller. For more information, visit http://www.cisco.com/en/US/ordering/index.shtml.

*Note* The part number for the Web services API optional license is CIAC-PAME-WSAPI=.  

**Step 3** When the purchase is complete, you are issued a Product Authorization Key (PAK) in paper form, or in an email message.

**Step 4** Continue to Installing the API License, page 1-10 for information on the two options used to download and install the license file using the PAK number.

Installing the API License

If your PC is connected to the Internet, you can enter the Product Authorization Key (PAK) to download and install a license file. You can also install a license file stored on a local disk.

This section includes the following information:
- Option 1: Enter the Product Authorization Key to Download the License File, page 1-10
- Option 2: Obtain the License File from the Cisco Web Site, page 1-11

**Option 1: Enter the Product Authorization Key to Download the License File**

*Note* To use this method, your PC must be connected to the Internet.

**Step 1** Locate the Product Authorization Key (PAK) created with the purchase of the optional feature.

**Step 2** Log on to the Cisco PAM appliance. See the Cisco Physical Access Manager User Guide for more information.

**Step 3** In the Setup List pane, choose License, as shown in Figure 1-3.

**Step 4** Enter the PAK code.

**Step 5** Click Update to download and install the license file on the appliance and activate the features.
Chapter 1      Overview

Enabling Web Services on the Cisco PAM Server

Figure 1-3       Installing Optional Feature Licenses

Note If the license file does not download, verify that your PC has Internet access, or use the method described in Option 2: Obtain the License File from the Cisco Web Site, page 1-11.

Step 6 Click the Features tab to verify that the new license was added. See Verifying the Installed Licenses, page 1-12 for more information.

Option 2: Obtain the License File from the Cisco Web Site

To use this method, obtain the license file from the Cisco Web site using a PC connected to the Internet, and transfer the file to the workstation used for server configuration.

Step 1 Locate the Product Authorization Key (PAK) created with the purchase of the optional feature.
Step 2 In a Web browser, open the Cisco Product License Registration Web page.
   http://www.cisco.com/go/license/
Step 3 Follow the on-screen instructions to complete the form and enter the Product Authorization Key (PAK). When you are done, a license file with the extension .lic is sent to your e-mail address.
Step 4 Transfer the file to the drive of the PC used for the configuration.
Step 5 In the License screen (Figure 1-3 on page 1-11), click Browse to select the license file located on your local drive. When selected, the file name appears in the File text box.
Step 6 Click Update to install the license file on the Cisco PAM appliance and activate the features.
Step 7 Click the Features tab to verify that the new license was added. See Verifying the Installed Licenses, page 1-12 for more information.
Verifying the Installed Licenses

From the Cisco PAM Server Administration utility, do the following:

**Step 1**  
In the Setup List pane, choose **License**, as shown in Figure 1-4.

**Step 2**  
Click the Features tab to view the installed licenses.

*Figure 1-4  License Features List*

Displaying the Cisco PAM Appliance Serial Number

To view the appliance serial number, do the following:

**Step 1**  
Log on to the Cisco PAM Server Administration utility. See the *Cisco Physical Access Manager User Guide*, or ask your system administrator for assistance.

**Step 2**  
Click the Monitoring tab, and then click **Server Status**, as shown in Figure 1-5.

**Step 3**  
Refer to the entry for **Server Serial Number**.
Authentication and Authorization

Before a method is called, use the authenticateUser API to send the Cisco PAM username and password, and retrieve a security context object (secCtx). Each subsequent API call uses this secCtx object as a parameter to authorize the API action.

**Note**

For the current Cisco PAM release, we recommend using the Administrator username and password for API authentication.

**Ending an API session**

API sessions end after a default idle time of 10 hours, or you can manually end the session using the logoutUser API.

- If a application session remains idle for a default duration of 10 hours, the session automatically ends and the security context object is deactivated.
- Applications can also use the logoutUser API to end an API session and deactivate the security context object.
- API calls using an expired security context object return a fault. See Chapter 3, “Fault Codes” for more information.
- Cisco PAM Web Service performs an automatic check every 10 minutes for idle sessions to expire.

**Note**

To begin a new API session, you must retrieve a new security context object (using authenticateUser).

**API Username and Password**

For the current Cisco PAM release, we recommend using the Administrator username and password for API authentication.

Usernames and passwords can also be configured using the Cisco PAM application to limit the API functionality:

- Use the Logins module to create the username and password for Cisco PAM client access.
- Use the Profiles module to define the Cisco PAM modules and commands available to a user.
For example, if an API application or user needs to view devices and events, the Logins username must be assigned a Profile with privileges to view events and devices. If an API user or application will invoke door commands, the username must include a profile with those privileges.

Tip
See the Cisco Physical Access Manager User Guide for instructions to configure Cisco PAM logins and profiles.

API Security

The Cisco PAM server and API use SSL for secure communication between the server and clients. The server uses a X.509 certificate (also called an SSL certificate) to verify its identity when a client attempts to connect to the server.

By default, the Cisco PAM server provides a self-signed certificate, which a client typically rejects. To prevent a client from rejecting this certificate, take one of the actions that Table 1-2 describes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
</table>
| If you are using a Java client, configure the SSL libraries for your clients to trust the self-signed certificate by using the Java keytool to import the certificate into the client truststore. | Procedure:  
1. SSH to the Cisco PAM server using the user cpamadmin.  
2. Enter sudo su to get a privileged shell.  
3. cpamservercert.jks has a Cisco PAM server certificate in the Java keystore. Use SFTP to copy this file on a client machine and import this certificate to the client trust store. |
| If you are using a client other than Java, configure the SSL libraries for your clients to trust the self-signed certificate. | You may be able to copy the self-signed certificate to a special directory on the client system. See your library documentation for detailed information. |
| Modify the way in which your client code validates certificate trust chains. | Some languages provide configurable SSL files that let you change the default certificate validation behavior. |

A client verifies its identity with a user name and password that are sent to the server by the client application.

Understanding Unique IDs

Many parameters include a user-defined ID number, and a machine generated unique ID.

- Readable IDs identify a specific record for an object, such as personId or badgeId. For example: a personId might be 3215. In some cases, this readable ID is used in the API request.
A unique ID (unid) for an object is used in most API requests, and is displayed in the API result. The unique ID for an object (such as a person or badge) is generated by the database and is the unique identifier for any record. For example, the unid for a person object might be
Z4JT5umCTzyCmVfvI6RAKw==.

Review the description for each API to determine which ID type is required.

API Logging

For debugging, API request and response messages are logged in the catalina log file located on the Cisco PAM server in the folder:
/opt/cisco/cpam/apache-tomcat/logs

By default, Web Service debug logs are written in webapp.log which is located in the folder:
/opt/cisco/cpam/logs
API Functions

This chapter provides detailed descriptions for each function in the Cisco Physical Access Control API. It includes these topics:

- Function Summary, page 2-1
- API Functions, page 2-6

Function Summary

Table 2-1 provides a summary of the Cisco Physical Access Control API functions. Each function is described in detail in the section that is listed.

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functions for Authentication</strong></td>
<td></td>
</tr>
<tr>
<td>authenticateUser, page 2-7</td>
<td>Authenticates the username and password and returns a security context object which must be passed in all subsequent API calls. This method must be called before any other method is called.</td>
</tr>
<tr>
<td>logoutUser, page 2-7</td>
<td>Ends an authenticated session. The security context object is expired and a new object must be generated to start a new API session.</td>
</tr>
<tr>
<td><strong>Physical Security Integration Management (PSIM)</strong></td>
<td></td>
</tr>
<tr>
<td>getAcDeviceTypes, page 2-8</td>
<td>Returns a list of physical access control device types, including the deviceTypeId, name, and a list of the possible device states.</td>
</tr>
<tr>
<td>getEventTypes, page 2-8</td>
<td>This method returns a list of event types defined in Cisco PAM. Each event type indicates event type unique ID, name and category of event.</td>
</tr>
<tr>
<td>getAllEventsCategories, page 2-9</td>
<td>Returns a list of the Cisco PAM event categories. Event categories can be used to call all the events that occurred for the event types that belong to that category.</td>
</tr>
<tr>
<td>getAllCameraDevices, page 2-10</td>
<td>Returns a list of the camera devices in Cisco PAM. Cameras are associated with one or more access control devices, such as a door.</td>
</tr>
<tr>
<td>Function Name and Reference</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getAllAcDevices, page 2-12</td>
<td>Returns a list of all access control devices configured in Cisco PAM.</td>
</tr>
<tr>
<td>getDeviceChildren, page 2-13</td>
<td>Returns the child devices of the specified device.</td>
</tr>
<tr>
<td>setDeviceAdminState, page 2-14</td>
<td>Changes the device admin state.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> This API supports door and reader devices only.</td>
</tr>
<tr>
<td>getAllDoorGroups, page 2-15</td>
<td>Returns the names of all door groups, including the unique ID for each group name.</td>
</tr>
<tr>
<td>getAllDoorsByGroup, page 2-15</td>
<td>Returns all doors configured for the specified door group.</td>
</tr>
<tr>
<td>getAllPerson, page 2-16</td>
<td>Returns the first name, last name, person ID and unique ID for access control personnel configured in Cisco PAM.</td>
</tr>
<tr>
<td>getAllPersonTypeIds, page 2-17</td>
<td>Returns all the personnel types defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAllPersonStatusIds, page 2-17</td>
<td>Returns a list of Status definitions for personnel in Cisco PAM.</td>
</tr>
<tr>
<td>getAllPersonByType, page 2-18</td>
<td>Returns the personnel records for a specified personnel type.</td>
</tr>
<tr>
<td>enrollPerson, page 2-18</td>
<td>Adds a personnel record to Cisco PAM.</td>
</tr>
<tr>
<td>updatePerson, page 2-19</td>
<td>Updates a person’s record in the Cisco PAM database. All properties in Cisco PAM are replaced by the new properties.</td>
</tr>
<tr>
<td>getPersonInfoByPersonId, page 2-20</td>
<td>Returns information for a personnel record, including a picture if available.</td>
</tr>
<tr>
<td>getCustomFieldsByPersonnel, page 2-21</td>
<td>Returns detailed information for the organizations defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAllDepartments, page 2-22</td>
<td>Returns all the departments defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAllEventsByCategory, page 2-22</td>
<td>Lists the events and alarms defined by the selected parameters, including event category, time interval, and the unique ID for the device.</td>
</tr>
<tr>
<td>getAllEventsByType, page 2-23</td>
<td>Lists events and alarms defined by the selected parameters, including event type, time interval, and the device unique ID.</td>
</tr>
<tr>
<td>getUserAccessEvents, page 2-25</td>
<td>Returns door access events related to granting or denying user access. Events can be returned for a specific user or device, or for all users and devices.</td>
</tr>
<tr>
<td>getAccessEventsByDoor, page 2-26</td>
<td>Returns access events and alarms related for a door.</td>
</tr>
<tr>
<td>getAllAlarmsByCategory, page 2-28</td>
<td>Lists alarms filtered by event category, time interval, and device ID.</td>
</tr>
<tr>
<td>getAllAlarmsByType, page 2-29</td>
<td>Lists alarms filtered by event type, time interval, and device ID.</td>
</tr>
<tr>
<td>getAllAlarmUpdates, page 2-30</td>
<td>Retrieves all alarm updates by getting a list of alarm annotations.</td>
</tr>
<tr>
<td>getAlarmInstructions, page 2-31</td>
<td>Returns information on Alarm instruction.</td>
</tr>
</tbody>
</table>
### Table 2-1  
**API Function Summary (continued)**

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearAlarm, page 2-32</td>
<td>Clears a specified alarm.</td>
</tr>
<tr>
<td>ackAlarm, page 2-34</td>
<td>Acknowledges an alarm.</td>
</tr>
<tr>
<td>getVersionInfo, page 2-34</td>
<td>Returns access control web service version information.</td>
</tr>
<tr>
<td>getAlarmDuplicates, page 2-35</td>
<td>Returns duplicates for a specified alarm.</td>
</tr>
</tbody>
</table>

**Event Notification APIs**

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registerAcEventCallback, page 2-36</td>
<td>Registers an event callback endpoint reference. Using this callback, events or alarms are notified to the endpoint referred by the endpoint reference.</td>
</tr>
<tr>
<td>registerAcEventXMLCallback, page 2-37</td>
<td>Registers an event callback endpoint URL. Using this callback events or alarms are notified to the endpoint pointed to by the service URL.</td>
</tr>
<tr>
<td>unregisterAcEventCallback, page 2-38</td>
<td>Unregisters an event callback. This is called when application is no longer interested getting notifications.</td>
</tr>
<tr>
<td>notifyAcEvent, page 2-39</td>
<td>Notifies a client application that registered a notification callback when an event or alarm occurs.</td>
</tr>
</tbody>
</table>

**Door Commands**

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantDoorAccess, page 2-41</td>
<td>Grants access to a specified door using the Grant Door Access command.</td>
</tr>
<tr>
<td>setDoorMode, page 2-41</td>
<td>Sets the door mode for a specified door and overrides the current door mode.</td>
</tr>
<tr>
<td>resetDoorMode, page 2-41</td>
<td>Resets the door mode to the original Cisco PAM configuration.</td>
</tr>
<tr>
<td>getDoorMode, page 2-42</td>
<td>Returns the current door mode for a specified door.</td>
</tr>
<tr>
<td>setDoorModeConfig, page 2-42</td>
<td>Sets the door mode, door configuration and applies any outstanding configuration changes from Cisco PAM.</td>
</tr>
</tbody>
</table>
| applyScheduleToDoor, page 2-43 | Applies the schedule to specified doors.  
**Note** Prior to this API, call the accessor API’s for the schedule and the door to retrieve database unique ids. |
| getAssociatedGWDoor, page 2-44 | Returns a list of associated devices [Door, GW] for the specified logical device Generic reader name in Cisco PAM. |

**Functions for Access Level and Schedule**

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllAccessLevels, page 2-44</td>
<td>Returns detailed information for all access levels defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAllAccessLevelNames, page 2-45</td>
<td>Returns names of all the access level defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAccessLevelByName, page 2-45</td>
<td>Returns detailed information for a specified access level.</td>
</tr>
<tr>
<td>getAllScheduleNames, page 2-46</td>
<td>Returns names of all the access schedules defined in Cisco PAM.</td>
</tr>
</tbody>
</table>
Table 2-1  API Function Summary (continued)

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getScheduleByName, page 2-47</td>
<td>Returns the schedule name with corresponding value for a specified schedule</td>
</tr>
<tr>
<td>enrollBadge, page 2-47</td>
<td>Adds one or more new badge credentials to an existing personnel record, or a new personnel record. If a new personnel record is specified, a picture in JPEG format can be included. Only new badges can be added using this API.</td>
</tr>
<tr>
<td>assignAccessLevels, page 2-49</td>
<td>Assigns one or more access levels to a badge associated with a person. <strong>Note</strong> This API replaces all the access levels on the specified badge.</td>
</tr>
<tr>
<td>getPersonInfoByBadge, page 2-49</td>
<td>Returns information for a personnel record for a given badge.</td>
</tr>
<tr>
<td>getAllBadgesByPerson, page 2-50</td>
<td>Returns badge details for the badges assigned to a specified person.</td>
</tr>
<tr>
<td>getBadgeValidityTypes, page 2-51</td>
<td>Returns a list of Cisco PAM badge validity types.</td>
</tr>
<tr>
<td>getCredentialTemplateName, page 2-51</td>
<td>Returns a list of names of Credential templates defined in CPAM</td>
</tr>
<tr>
<td>getBadgeRoles, page 2-52</td>
<td>Returns the Badge type (roles) defined in Cisco PAM.</td>
</tr>
<tr>
<td>getAllBadgeTemplateName, page 2-52</td>
<td>Returns the Badge Template names defined in Cisco PAM. The names are used as a parameter in the enrollBadge API.</td>
</tr>
<tr>
<td>getAccessLevelByPerson, page 2-53</td>
<td>Returns a list of the access levels for all badges assigned to a person. The person is identified by a unique ID.</td>
</tr>
<tr>
<td>getBadgeByCardNum, page 2-53</td>
<td>Retrieves a badge record for a specified card number.</td>
</tr>
<tr>
<td>updateBadge, page 2-54</td>
<td>Updates a list of badge credential in the Cisco PAM database. All properties on a badge in Cisco PAM are replaced by the new properties for the specified badge.</td>
</tr>
<tr>
<td>getAllUpdatedBadges, page 2-56</td>
<td>Returns all the badges that are either modified or created between beginTime and now.</td>
</tr>
<tr>
<td>getAllUpdatedPersonnels, page 2-56</td>
<td>Returns details of all personnel (the first name, last name, user ID and unique ID) that are either modified or created between beginTime and now.</td>
</tr>
<tr>
<td>getAccessLevelByUnid, page 2-57</td>
<td>Returns information about an access level specified by the access level unique id.</td>
</tr>
<tr>
<td>getAllBadgeTypeIds, page 2-58</td>
<td>Returns the unique identifiers for Cisco PAM badge types.</td>
</tr>
<tr>
<td>getAllBadgeRefs, page 2-58</td>
<td>Returns a range of badge references based on required beginning and ending card number parameters.</td>
</tr>
</tbody>
</table>

**Functions for Time and Schedule based on Location**

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createTimeEntry, page 2-60</td>
<td>Creates a time entry record and returns a database uniqueId.</td>
</tr>
<tr>
<td>updateTimeEntry, page 2-60</td>
<td>Updates the time entry record with the uniqueId as specified in the timeEntry argument.</td>
</tr>
<tr>
<td>deleteTimeEntry, page 2-61</td>
<td>Deletes the time entry record referenced by the uniqueId.</td>
</tr>
</tbody>
</table>
### Table 2-1  API Function Summary (continued)

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllTimeEntries, page 2-61</td>
<td>List all the TimeEntries under the scope of the specified location.</td>
</tr>
<tr>
<td>getTimeEntryById, page 2-63</td>
<td>Returns the TimeEntry based on the specified uniqueId.</td>
</tr>
<tr>
<td>getTimeEntryByName, page 2-63</td>
<td>Returns the TimeEntry based on the specified name.</td>
</tr>
<tr>
<td>getAllSchedulesForTimeEntry, page 2-65</td>
<td>Returns all schedules based on the specified time entry. This API supports time entries of the type Holiday and Workweek only.</td>
</tr>
<tr>
<td>createTimeRangeGroup, page 2-65</td>
<td>Creates a time range record and returns a database uniqueId.</td>
</tr>
<tr>
<td>updateTimeRangeGroup, page 2-67</td>
<td>Updates the time range record with the uniqueId as that of the specified argument.</td>
</tr>
<tr>
<td>deleteTimeRangeGroup, page 2-67</td>
<td>Deletes the time range record identified by the uniqueId.</td>
</tr>
<tr>
<td>getAllTimeRangeGroups, page 2-67</td>
<td>List all TimeRanges under the scope of the specified location.</td>
</tr>
<tr>
<td>getTimeRangeGroupById, page 2-69</td>
<td>Returns time range group based on the specified uniqueId.</td>
</tr>
<tr>
<td>getTimeRangeGroupByName, page 2-69</td>
<td>Returns time range group based on the specified name.</td>
</tr>
<tr>
<td>createTimeEntryCollection, page 2-69</td>
<td>Creates a time entry collection as specified in the input argument and returns a unique database ID.</td>
</tr>
<tr>
<td>updateTimeEntryCollection, page 2-71</td>
<td>Updates the time entry collection as specified in the input argument.</td>
</tr>
<tr>
<td>deleteTimeEntryCollection, page 2-71</td>
<td>Deletes the time entry collection record identified by the uniqueId.</td>
</tr>
<tr>
<td>getAllTimeEntryCollections, page 2-72</td>
<td>Lists all TimeEntryCollections under the scope of the specified location.</td>
</tr>
<tr>
<td>getTimeEntryCollectionById, page 2-72</td>
<td>Returns time entry collection based on the specified uniqueId.</td>
</tr>
<tr>
<td>getTimeEntryCollectionByName, page 2-73</td>
<td>Returns time entry collection based on the specified name.</td>
</tr>
<tr>
<td>getAllSchedulesForTimeEntryCollection, page 2-73</td>
<td>Returns all schedules based on the specified time entry collection.</td>
</tr>
<tr>
<td>createSchedule, page 2-75</td>
<td>Creates a schedule as specified in the input argument and returns a unique database ID.</td>
</tr>
<tr>
<td>updateSchedule, page 2-75</td>
<td>Updates the schedule in the specified argument.</td>
</tr>
<tr>
<td>addExceptionToSchedule, page 2-77</td>
<td>Adds the provided schedule entry as an exception in the specified schedule.</td>
</tr>
<tr>
<td>deleteSchedule, page 2-77</td>
<td>Deletes the schedule record identified by the uniqueId of the input argument along with the schedule entry(ies).</td>
</tr>
<tr>
<td>getAllSchedules, page 2-78</td>
<td>Lists all the schedules under the scope of the specified location.</td>
</tr>
<tr>
<td>getAllDoorSchedules, page 2-78</td>
<td>Lists all the door schedule definitions under the scope of the specified location.</td>
</tr>
</tbody>
</table>
### Table 2-1  API Function Summary (continued)

<table>
<thead>
<tr>
<th>Function Name and Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllSchedulesForTimeRangeGroup, page 2-80</td>
<td>Returns all schedules based on the specified time range.</td>
</tr>
<tr>
<td><strong>Location Listing APIs</strong></td>
<td></td>
</tr>
<tr>
<td>getAllLocationsInScope, page 2-81</td>
<td>Returns a list of locations in the sub-tree of the logged-in user’s default location.</td>
</tr>
<tr>
<td>getAncestorLocations, page 2-81</td>
<td>Returns all the locations leading to the logged in user’s location starting from the root of the location hierarchy.</td>
</tr>
<tr>
<td><strong>Function for Configuration Changes</strong></td>
<td></td>
</tr>
<tr>
<td>applyConfigurationChanges, page 2-83</td>
<td>Returns information on the changes to gateways in the logged in user’s location scope.</td>
</tr>
<tr>
<td>applyConfigurationChangesToLocation, page 2-83</td>
<td>Pushes the configuration changes to gateways in the specified location.</td>
</tr>
<tr>
<td><strong>Recording External Events</strong></td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>This section also includes instructions to define the events used the Cisco PAM External Events module. See Recording External Events, page 2-85.</td>
</tr>
<tr>
<td>recordExtEvent, page 2-89</td>
<td>Records an external event or alarm in Cisco PAM.</td>
</tr>
</tbody>
</table>

### API Functions

The following sections describe each Cisco Physical Access Control API function in detail. The function descriptions provide the following information:

- Function name—Name of the function.
- Description—General behavior for the function.
- Parameters—Parameters that the function requires.
- Sample Request—Example of the API request.
- Return result—Describes the result for the function.
- Sample response—Example of a response that the function can receive.

**Note**

- The Method for all Cisco PAM APIs is POST.
- Sample request and response payloads are provided for each API. APIs can use either SOAP/HTTP or XML/HTTP. The sample requests and responses are for illustration purposes only.
## Authentication APIs

### authenticateUser

<table>
<thead>
<tr>
<th>Function name</th>
<th>authenticateUser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Authenticates the username and password and returns a security context object which must be passed in all subsequent API calls. This method must be called before any other method is called.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>username—(required) unique Cisco PAM log in ID.</td>
</tr>
<tr>
<td></td>
<td>password—(required) Cisco PAM user password.</td>
</tr>
</tbody>
</table>
<username>cpamadmin</username>
<password>cpamadmin</password>
</ns2:authenticateUser>` |
| Return result | A security token is returned if authentication is successful. The token references a security context object maintained on the Cisco PAM API service. This object also maintains user session information. The token number must be provided using the `secCtx` parameter in each subsequent API call. |

### logoutUser

<table>
<thead>
<tr>
<th>Function name</th>
<th>logoutUser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Ends an authenticated session. The security context object is expired and a new object must be generated to start a new API session.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
</tbody>
</table>
<in token="6193613" />
</ns2:logoutUser>` |
| Return result | All notification listeners registered in the user session are removed. Users must log back in using authenticateUser to retrieve a new security context object before executing additional APIs. |
Physical Security Integration Management (PSIM) APIs

getAcDeviceTypes

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAcDeviceTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns a list of physical access control device types, including the deviceTypeId, name, and a list of the possible device states.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>Sample request</td>
<td>&lt;ns2:getAcDeviceTypes xmlns:ns3=&quot;<a href="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing</a>&quot; xmlns:ns2=&quot;<a href="http://cisco.com/physec/acws/">http://cisco.com/physec/acws/</a>&quot; secCtx token=&quot;6193613&quot; /&gt;</td>
</tr>
<tr>
<td>Return result</td>
<td>A list of the physical access control device types, including the type ID, name, and possible device states.</td>
</tr>
<tr>
<td>Sample response</td>
<td></td>
</tr>
</tbody>
</table>

getEventTypes

<table>
<thead>
<tr>
<th>Function name</th>
<th>getEventTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This method returns a list of event types defined in Cisco PAM. Each event type indicates event type unique ID, name and category of event.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
getAllEventsCategories

Function name | getAllEventsCategories
---|---
Description | Returns a list of the Cisco PAM event categories. Event categories can be used to call all the events that occurred for the event types that belong to that category.
Method | Post
Parameters | • secCtx—(required) the security context object returned by authenticateUser.
<secCtx token="7163697" />
</ns2:getAllEventCategories>
## getAllCameraDevices

### Function name
getAllCameraDevices

### Description
Returns a list of the camera devices in Cisco PAM. Cameras can be associated with one or more access control devices, such as a door.

### Method
Post

### Parameters
- `secCtx`—(required) the security context object returned by `authenticateUser`.

### Sample request
```xml
<ns2:getAllCameraDevices xmlns:ns2="http://cisco.com/physec/acws/"
                           xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="1826753" />
</ns2:getAllCameraDevices>
```
Return result | A list of camera devices with unique ID, name, type and list of associated devices.
--- | ---
Sample response | <ns2:getAllCameraDevicesResponse
xmlns:ns2="http://cisco.com/physec/acws/
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<cameras>
    <devices xsi:type="ns2:CameraDevice" type="CB.CAMERA" name="S1 Cam Lab 1" unid="2ZuPNca/QMeVjV19KEj1PA==" />
    <devices xsi:type="ns2:CameraDevice" type="CB.CAMERA" name="Panasonic1_new" unid="9cKGFhFPTdKxewHj10Iwmg==" >
    </devices>
    <devices xsi:type="ns2:AcDoorDevice" enabled="true" siteId="DsL5KGP8Tp+w2s/oN4C5g==" deviceState="GW.UP" location="Cisco Iodc/Building 9/First floor" address="/SanJose/pgw_123/Door/door00" type="GW.DOOR" name="door00" unid="Tu9N06ZcSi4HIC4f4pGw==" >
        <cameras>
            <devices xsi:type="ns5:CameraDevice" type="CB.CAMERA" name="Panasonic1_new" unid="9cKGFhFPTdKxewHj10Iwmg==" />
            <devices xsi:type="ns5:CameraDevice" type="CB.CAMERA" name="Panasonic_1" unid="e/ZR1B/ISVfKp431q5Gw==" />
        </cameras>
        <adminStatus>ADMINUP</adminStatus>
        <associatedDevice enabled="true" siteId="DsL5KGP8Tp+w2s/oN4C5g==" deviceState="GW.UP" address="/SanJose/pgw_123/m00/input01/rex" type="GW.REX" name="rex" unid="/cUCkOHIRwKGlXZBK4A5Q==" >
            <adminStatus>ADMINUP</adminStatus>
        </associatedDevice>
        <associatedDevice enabled="true" siteId="DsL5KGP8Tp+w2s/oN4C5g==" deviceState="GW.UP" address="/SanJose/pgw_123/m00/rdr01/reader" type="GW.RDR" name="reader" unid="wPfDHYC7SiuLaIZgr7S1GY2==" >
            <adminStatus>ADMINUP</adminStatus>
        </associatedDevice>
        <associatedDevice enabled="true" siteId="DsL5KGP8Tp+w2s/oN4C5g==" deviceState="GW.UP" address="/SanJose/pgw_123/m00/input02/doorSensor" type="GW.DOORSENSOR" name="doorSensor" unid="Dmpm7RP0QfmCbj9+2Kfm5w==" >
            <adminStatus>ADMINUP</adminStatus>
        </associatedDevice>
        <associatedDevice enabled="true" siteId="DsL5KGP8Tp+w2s/oN4C5g==" deviceState="GW.UP" address="/SanJose/pgw_123/m00/output02/lock" type="GW.LOCK" name="lock" unid="UU14orvoR+aG749S18S5Q==" >
            <adminStatus>ADMINUP</adminStatus>
        </associatedDevice>
    </devices>
</cameras>
</ns2:getAllCameraDevicesResponse>
### getAllAcDevices

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllAcDevices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns a list of all access control devices configured in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters      | • `secCtx`—(required) the security context object returned by `authenticateUser`.  
• `dtypes`—(optional) a collection of device types. Only the access control devices of the specified device types will be returned in the result. If not specified, all access control devices types are returned. |
| Sample request  | In the following XML sample request, only door devices are requested.  
```
  <secCtx token="1826753" />
  <dtypes>
    <dtype deviceTypeName="Door" deviceTypeId="GW.DOOR" />
  </dtypes>
</ns2:getAllAcDevices>
``` |
getDeviceChildren

<table>
<thead>
<tr>
<th>Function name</th>
<th>getDeviceChildren</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the child devices of the specified device.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>

Sample response

```xml
  <devices>
    <devices xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:AcDoorDevice" enabled="true" siteId="v5OyH3XSqQnRECjKcAsQ==" deviceState="VX.UNK" address="/SanJose/pgw_1001/Door/door00" type="GW.DOOR" name="FrontDoor" unid="XMT6EusqSaaSBeM78eFQkwen=">
      <cameras>
        <devices xmlns:ns5="http://cisco.com/physec/video/" xsi:type="ns5:CameraDevice" type="CB.CAMERA" name="2500-SD-Dual" unid="CquVLW6RZ0LYos6xfMUrA==" />
        <devices xmlns:ns5="http://cisco.com/physec/video/" xsi:type="ns5:CameraDevice" type="CB.CAMERA" name="702 Xp3 0" unid="8evRj5pOTaSxT7D0N9vIqe=" />
      </cameras>
      <adminStatus>PROVISIONED</adminStatus>
      <associatedDevice enabled="true" siteId="v5OyH3XSqQnRECjKcAsQ==" deviceState="VX.UNK" address="/SanJose/pgw_1001/m00/input01/rex" type="GW.REX" name="rex" unid="cEWC1cqmRsW7v3KdfSp2g==" />
      <adminStatus>PROVISIONED</adminStatus>
      <associatedDevice enabled="true" siteId="v5OyH3XSqQnRECjKcAsQ==" deviceState="VX.UNK" address="/SanJose/pgw_1001/m00/input02/doorSensor" type="GW.DOORSENSOR" name="doorSensor" unid="Z1Lj4u1CTDXYS3ToAbrg8w==" />
      <adminStatus>PROVISIONED</adminStatus>
      <associatedDevice enabled="true" siteId="v5OyH3XSqQnRECjKcAsQ==" deviceState="VX.UNK" address="/SanJose/pgw_1001/m00/rdr01/reader" type="GW.RDR" name="reader" unid="pxLjvv8UR57D7Dz+vJ9xZw==" />
      <adminStatus>PROVISIONED</adminStatus>
      <associatedDevice enabled="true" siteId="v5OyH3XSqQnRECjKcAsQ==" deviceState="VX.UNK" address="/SanJose/pgw_1001/m00/output01/lock" type="GW.LOCK" name="lock" unid="5+knHFL2Q/6P+77VF8clEA==" />
      <adminStatus>PROVISIONED</adminStatus>
    </associatedDevice>
  </devices>
</ns2:getAllAcDevicesResponse>
### setDeviceAdminState

<table>
<thead>
<tr>
<th>Function name</th>
<th>setDeviceAdminState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Change the device admin state.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>secCtx— the security context object returned by authenticateUser.</td>
<td></td>
</tr>
<tr>
<td>deviceId— specifies the device unique id.</td>
<td></td>
</tr>
<tr>
<td>adminState— specifies the state to be changed for the given device. The values are [ADMINUP, ADMINDOWN].</td>
<td></td>
</tr>
</tbody>
</table>

**Sample request**

```xml
 xmlns:ns3="http://www.w3.org/2005/08/addressing"
<secCtx token="4316032"/>
<deviceId>X8cUx8BtRGGv40cox+HL+w==</deviceId>
<adminState>ADMINDOWN</adminState>
</ns2:setDeviceAdminState>
```

**Return result**

Response indicates that device state has changed successfully.

**Sample response**

```xml
 xmlns:ns3="http://www.w3.org/2005/08/addressing"/>
```
### getAllDoorGroups

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllDoorGroups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the names of all door groups, including the unique ID for each group name.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>• <code>secCtx</code>—(required) the security context object returned by <code>authenticateUser</code>.</td>
</tr>
</tbody>
</table>

**Sample request**

```xml
<ns2:getAllDoorGroups xmlns:ns3="http://www.w3.org/2005/08/addressing"
ns2="http://cisco.com/physec/acws/">
  <secCtx token="9440512" />
</ns2:getAllDoorGroups>
```

**Return result**
The names and unique ID of all door groups configured in Cisco PAM.

**Sample response**

```xml
<ns2:getAllDoorGroupsResponse xmlns:ns3="http://www.w3.org/2005/08/addressing"
ns2="http://cisco.com/physec/acws/">
  <doorGroups name="Cube-doors" unid="0LYtFLOKS1SfM0kpnpIlew=" />
</ns2:getAllDoorGroupsResponse>
```

### getAllDoorsByGroup

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllDoorsByGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns all doors configured for the specified door group.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters          | • `secCtx`—(required) the security context object returned by `authenticateUser`.  
• `groupId`—(required) unique ID of the door group. |

**Sample request**

```xml
<ns2:getAllDoorsByGroup xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="29223376" />
  <groupId>0LYtFLOKS1SfM0kpnpIlew==</groupId>
</ns2:getAllDoorsByGroup>
```
### getAllPerson

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllPerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the first name, last name, person ID and unique ID for access control personnel configured in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
### API Functions

#### getAllPersonTypeIds

**Function name**  
getAllPersonTypeIds

**Description**  
Returns all the personnel types defined in Cisco PAM.

**Method**  
Post

**Parameters**  
- `secCtx` — (required) the security context object returned by `authenticateUser`.

**Sample request**

```xml
<ns2:getAllPersonTypeIds xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="23012302"/>
</ns2:getAllPersonTypeIds>
```

**Return result**  
List of the personnel types.

To view the list in Cisco PAM:

1. In the Users list, click **Personnel**.
2. Open a personnel record and click the Occupational Information tab.
3. Click the **Personnel types** menu to display the list.

**Sample response**

```xml
<ns2:getAllPersonTypeIdsResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <personType>Contractor</personType>
  <personType>Employee</personType>
  <personType>Employee - Full Time</personType>
  <personType>Employee - Part Time</personType>
  <personType>Intern</personType>
  <personType>XXXX</personType>
  <personType>Other</personType>
</ns2:getAllPersonTypeIdsResponse>
```

#### getAllPersonStatusIds

**Function name**  
getAllPersonStatusIds

**Description**  
Returns a list of Status definitions for personnel in Cisco PAM.

**Method**  
Post

**Parameters**  
- `secCtx` — (required) the security context object returned by `authenticateUser`.

**Sample request**

```xml
<ns2:getAllPersonStatusId xmlns:ns2="http://cisco.com/physec/acws/"
 xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="32015437"/>
</ns2:getAllPersonStatusId>
```

**Return result**  
A list of the access control persons, including first name, last name, person ID and unique ID.

**Sample response**

```xml
<ns2:getAllPersonStatusIdResponse xmlns:ns2="http://cisco.com/physec/acws/"
 xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <personStatus>Active</personStatus>
  <personStatus>Contractor</personStatus>
  <personStatus>Employee</personStatus>
  <personStatus>Employee - Full Time</personStatus>
  <personStatus>Employee - Part Time</personStatus>
  <personStatus>Intern</personStatus>
  <personStatus>Other</personStatus>
</ns2:getAllPersonStatusIdResponse>
```
### getAllPersonByType

**Function name**  getAllPersonByType  
**Description**  Returns the personnel records for a specified personnel type.  
**Method**  Post  
**Parameters**  
- `secCtx`—(required) the security context object returned by `authenticateUser`.  
- `typeId`—(required) one of the personnel types returned using the `getAllPersonTypeIds` API.  
**Sample request**  
```
<ns2:getAllPersonByType xmlns:ns3="http://www.w3.org/2005/08/addressing"
 xmlns:ns2="http://cisco.com/physec/acws/">
 <secCtx token="32015437" />
 <typeId>EMPLOYEE_FULL_TIME</typeId>
</ns2:getAllPersonByType>
```

**Return result**  Returns a list of personnel records for the specified personnel type.  
**Sample response**  
```
<ns2:getAllPersonByTypeResponse xmlns:ns3="http://www.w3.org/2005/08/addressing"
 xmlns:ns2="http://cisco.com/physec/acws/">
 <persons>
  <person lname="TestUser_LastName" fname="TestUser_FirstName"
   personId="123456" unid="wWyRKdm6QyuCNes9J3NXaA==" />
 </persons>
</ns2:getAllPersonByTypeResponse>
```

### enrollPerson

**Function name**  enrollPerson  
**Description**  Adds a personnel record to Cisco PAM.  
**Sample request**  
```
<ns2:getAllPersonStatusIds xmlns:ns3="http://www.w3.org/2005/08/addressing"
 xmlns:ns2="http://cisco.com/physec/acws/">
 <secCtx token="7163697" />
</ns2:getAllPersonStatusIds>
```

**Return result**  List of the Status definitions.  
To view the list in Cisco PAM:  
- In the Users list, click Personnel.  
- Open a personnel record and click the Occupational Information tab.  
- Click the Status menu to display the list.  
**Sample response**  
```
<ns2:getAllPersonStatusIdsResponse xmlns:ns3="http://www.w3.org/2005/08/addressing"
 xmlns:ns2="http://cisco.com/physec/acws/">
 <status>ACTIVE</status>
 <status>INACTIVE</status>
 <status>ON_LEAVE</status>
 <status>RETIRED</status>
 <status>TERMINATED</status>
</ns2:getAllPersonStatusIdsResponse>
```
## updatePerson

**Function name**  
updatePerson

**Description**  
Updates a person’s record in the Cisco PAM database. All properties in Cisco PAM are replaced by the new properties.

**Method**  
Post

**Parameters**

- `secCtx`—(required) the security context object returned by `authenticateUser`.
- `personInfo`—(required) data assigned to the personnel record. This can include the `personId`, `firstName`, `lastName`, `organizationId` (can be fetched from the `getCustomFieldsByPersonnel` API), `departmentId` (can be fetched from the `getAllDepartments` API), `statusId`, and `typeId`. The personnel record can also include a base 64 encoded JPG image of the person.

**Sample request**

```xml
  <secCtx token="4065879" />
  <pinfo statusId="ACTIVE" type="EMPLOYEE_FULL_TIME"
           departmentId="4MbV9fN5W67g71gB78jQ==" organizationId="kY1IGV6kTGCUFVOMucOMGg=="
           lname="Last_3" fname="First_3" personId="000000013" unid="xx4M7jMiRX2x/cZRsEANDg=" />
  ...
</ns2:updatePerson>
```

**Return result**  
If the person is enrolled successfully, the unique ID of the person is returned.

**Sample response**

```xml
  <personId>4444</personId>
</ns2:enrollPersonResponse>
```
### getPersonInfoByPersonId

<table>
<thead>
<tr>
<th>Function name</th>
<th>getPersonInfoByPersonId</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns information for a personnel record, custom fields, including a picture if available.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>secCtx—(required)</td>
<td>the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>personId—(required)</td>
<td>the unique ID of the person.</td>
</tr>
</tbody>
</table>

#### Sample request

```xml
<ns2:getPersonInfoByPersonId xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<secCtx token="21205590"/>
<personId>1yMtnz6gQyijLZ8+oGMoEg==</personId>
</ns2:getPersonInfoByPersonId>
```

#### Return result

Returns the information configured for the specified person (PersonInfo object). If the object includes the person’s picture, a base 64 encoded JPG image is also included.

#### Sample response

```xml
<ns2:getPersonInfoByPersonIdResponse xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<personInfo statusId="ACTIVE" lname="L_name" fname="F_name" personId="101"
unid="1yMtnz6gQyijLZ8+oGMoEg==">
<customextension value1="Custom Field2" value0="Custom Field1"/>
</personInfo>
</ns2:getPersonInfoByPersonIdResponse>
## getCustomFieldsByPersonnel

**Function name**

getCustomFieldsByPersonnel.

**Description**

Returns only Custom fields.

**Method**

Post

**Parameters**

- SecCtx: the security context object returned by authenticate User.
- personId—(required) the unique ID of the person.

**Sample request**

```xml
<ns2:getCustomFieldsByPersonnel xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="21283430"/>
  <personId>++4MIVTlQdSuDCFU5bHUhg==</personId>
</ns2:getCustomFieldsByPersonnel>
```

**Return result**

Returns the custom fields value associated with person Id.

**Sample response**

```xml
  <personInfo>
    <customextension value2="two " value1="one " value0="zero"/>
  </personInfo>
</ns2:getCustomFieldsByPersonnelResponse>
```

## getAllOrganizations

**Function name**

getAllOrganizations

**Description**

Returns detailed information for the organizations defined in Cisco PAM.

**Method**

Post

**Parameters**

- secCtx—(required) the security context object returned by authenticateUser.

**Sample request**

```xml
  <secCtx token="7163697"/>
</ns2:getAllOrganizations>
```

**Return result**

List of organizations in Cisco PAM, including the unique ID, name and a list of department IDs.

**Sample response**

```xml
  <org>
    <uniqueId>kYIGV6k9TGCUFVOMucOM0g==</uniqueId>
    <name>Engineering</name>
    <deptIds>4MbV9fvNSWGC71gu7B7SjQ==</deptIds>
  </org>
  <org>
    <uniqueId>zKhxXl7pSV6y1ZzP/7sGkg==</uniqueId>
    <name>Marketing</name>
    <deptIds>fmVl1ZqCTUSo7R7m6JRJSQ==</deptIds>
  </org>
</ns2:getAllOrganizationsResponse>
### getAllDepartments

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllDepartments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns all the departments defined in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
</tbody>
</table>

#### Sample request

```xml
<ns2:getAllDepartments xmlns:ns3="http://www.w3.org/2005/08/addressing" xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="7163697" />
</ns2:getAllDepartments>
```

#### Return result

List of the departments in Cisco PAM. A department has a unique ID, name and the ID of the organization that it is a part of.

#### Sample response

```xml
<ns2:getAllDepartmentsResponse xmlns:ns3="http://www.w3.org/2005/08/addressing" xmlns:ns2="http://cisco.com/physec/acws/">
  <dept>
    <uniqueId>4MbY9fvNSWGe7Igu7B7SjQ==</uniqueId>
    <name>PSBUAccessControl</name>
    <orgId>kYIGV6k9TGCUFVOMucOM0g==</orgId>
  </dept>
  <dept>
    <uniqueId>fmVolZqCTUSotR7m6JRJ5Q==</uniqueId>
    <name>VideoSurvellance</name>
    <orgId>zKhxXl7pSV6y1ZZP/7sGkg==</orgId>
  </dept>
</ns2:getAllDepartmentsResponse>
```

### getAllEventsByCategory

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllEventsByCategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lists the events and alarms defined by the selected parameters, including event category, time interval, and the unique ID for the device.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>category—(string: required): the event category.</td>
</tr>
<tr>
<td></td>
<td>beginTime—(long: optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.</td>
</tr>
<tr>
<td></td>
<td>endTime—(long: optional) the interval endTime. Ending time for a time interval, specified as number of milliseconds from epoch time.</td>
</tr>
<tr>
<td></td>
<td>deviceId—(optional) the unique ID of the device. If a unique ID is included, events and alarms are only returned for the specified device. If blank, events of all devices are included.</td>
</tr>
</tbody>
</table>
Sample request

```xml
<ns2:getAllEventsByCategory xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="11563709" />
  <category>GW.EV_MAJOR_ID_DOOR</category>
  <begTime>1244498633952</begTime>
  <endTime>1247090633952</endTime>
</ns2:getAllEventsByCategory>
```

Return result

Returns the events and alarms by event category and other optional parameters.

Sample response

```xml
<ns2:getAllEventsByCategoryResponse
  xmlns:ns2="http://cisco.com/physec/acws/">
  <events>
    <event evtTime="1228950257000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="126">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228950535000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="129">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228950633000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="133">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228955167000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="139">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1229014038000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="168">
      <description>Door Forced Open</description>
    </event>
  </events>
</ns2:getAllEventsByCategoryResponse>

**getAllEventsByType**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllEventsByType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lists events and alarms defined by the selected parameters, including event type, time interval, and the device unique ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
Parameters

- **secCtx**—(required) the security context object returned by authenticateUser.
- **evtTypeUnid**—(string: required) the unique ID of the event type.
- **beginTime**—(long: optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.
- **endTime**—(long: optional) the interval endTime. Ending time for a time interval, specified as number of milliseconds from epoch time.
- **deviceId**—(optional) the unique ID of the device. If blank, all devices are included. If a device ID is included, events and alarms are only returned for the specified device.

**Note** If an invalid evtTypeUnid or deviceId is specified, no matching events will be returned.

Sample request

The following XML example request specifies a category and time interval:

```xml
<ns2:getAllEventsByType xmlns:ns3="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="11563709" />
  <evtTypeUnid>GW.EV_DOOR_FORCED_OPEN</evtTypeUnid>
  <begTime>1244498636499</begTime>
  <endTime>1247090636499</endTime>
</ns2:getAllEventsByType>
```

Return result

Returns the events and alarms by event type and other optional parameters.

**Note** If an invalid evtTypeUnid or deviceId is specified, no matching events will be returned.

Sample response

```xml
<ns2:getAllEventsByTypeResponse
xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <events>
    <event evtTime="1228950257000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="126">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228950535000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="129">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228950633000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="133">
      <description>Door Forced Open</description>
    </event>
    <event evtTime="1228955167000" isAlarm="true"
      type="GW.EV_DOOR_FORCED_OPEN"
      deviceId="GgXIKOh2Rciu33wuB5jXdA=="
      unid="139">
      <description>Door Forced Open</description>
    </event>
  </events>
</ns2:getAllEventsByTypeResponse>
```
### getUserAccessEvents

<table>
<thead>
<tr>
<th>Function name</th>
<th>getUserAccessEvents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns door access events related to granting or denying user access. Events can be returned for a specific user or device, or for all users and devices.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | secCtx—(required) the security context object returned by authenticateUser.  
beginTime—(long: optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.  
endTime—(long: optional) the interval endTime. Ending time for a time interval, specified as number of milliseconds from epoch time.  
personId—(optional) the unique ID of an access control user. If not specified, door access events for all users are returned.  
deviceId—(optional) the unique ID of the door device. If a deviceId is not included, events for all door devices are returned. If a device unique ID is included, events and alarms are only returned for the specified device.  
max—(int: optional) specifies the maximum number of events.  
recent—(bool: optional) if true, the most recent events are returned first. If false or null, the oldest events are returned first. |
| Note | If an invalid personId or deviceId is specified, no matching events will be returned. |
| Sample request | In the following example, a time interval is specified, a maximum 1,000 events are included, and events are returned by most recent first:  
```xml
  <secCtx token="11198631"/>
  <begTime>1374040297918</begTime>
  <endTime>1376632297918</endTime>
  <personId>++4MIVTlQdSuDCFU5bHUhg==</personId>
  <maxCount>1000</maxCount>
  <recent>true</recent>
</ns2:getUserAccessEvents>
``` |
## Return result

Returns information for events in the GW.EV_MAJOR_ID_DOOR category, including the following:

- "EV_DOOR_GRANT_ACCESS"
- "EV_DOOR_GRANT_ACCESS_EXTENDED"
- "EV_DOOR_DENY_ACCESS_NO_ACC"
- "EV_DOOR_DENY_ACCESS_DOORSCHED"
- "EV_DOOR_DENY_ACCESS_WRONG_PIN"
- "EV_DOOR_DENY_ACCESS_TIMED_APB"
- "EV_DOOR_DENY_ACCESS_APB"
- "EV_DOOR_DENY_ACCESS_BLACKLIST"

Events are filtered by person, device, time interval and other optional parameters.

**Note**

If an invalid eventType or deviceId is specified, no matching events will be returned.

### Sample response

```xml
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<events>
<event isAnnotation="false" siteId="8bi7nX/mROy4u51/SiRs7g==" priority="0"
evTime="1376486184000" isAlarm="false" type="GW.EV_DOOR_GRANT_ACCESS"
deviceId="nttfFavTQhXWr06YIA==" unid="23601">
<description>Door Grant Access</description>
<accessEvtProps punId="++4MIVTlQdSuDCFU5bHUhg==" accessType="ENTRY_READER"
lname="Blokdijk" fname="Karsten" personId="40101610"/>
<srcDeviceName>d1_entry</srcDeviceName>
</event>
<event isAnnotation="false" siteId="8bi7nX/mROy4u51/SiRs7g==" priority="0"
evTime="1376486182000" isAlarm="false" type="GW.EV_DOOR_GRANT_ACCESS"
deviceId="Kme7/4jkRdGnu/+I5EXXvQ==" unid="23541">
<description>Door Grant Access</description>
<accessEvtProps punId="++4MIVTlQdSuDCFU5bHUhg==" accessType="EXIT_READER"
lname="Blokdijk" fname="Karsten" personId="40101610"/>
<srcDeviceName>D2_Exit</srcDeviceName>
</event>
</events>
</ns2:getUserAccessEventsResponse>
```

### getAccessEventsByDoor

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAccessEventsByDoor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns access events and alarms related for a door.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>beginTime—(long: optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.</td>
</tr>
<tr>
<td></td>
<td>endTime—(long: optional) the interval endTime. Ending time for a time interval, specified as number of milliseconds from epoch time.</td>
</tr>
<tr>
<td></td>
<td>deviceId—(required) the unique ID of the door device. Returns events and alarms for the specified device. If blank, a fault is returned (see Chapter 3, “Fault Codes”).</td>
</tr>
<tr>
<td></td>
<td>max—(int: optional) specifies the maximum number of events.</td>
</tr>
<tr>
<td></td>
<td>recent—(bool: optional) if true, the most recent events are returned first. If false or null, the oldest events are returned first.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample request</th>
<th>In the following example, a time interval is specified, a maximum 1,000 events are included, and events are returned by most recent first:</th>
</tr>
</thead>
</table>
|               |     <secCtx token="12272293" />
|               |     <beginTime>1234746808724</beginTime>
|               |     <endTime>1237338808724</endTime>
|               |     <deviceId>foGrYqOjT7yuxPDOw6/9Yg==</deviceId>
|               |     <maxCount>5</maxCount>
|               |     <recent>true</recent>
|               | </ns2:getAccessEventsByDoor> |
### API Functions

<table>
<thead>
<tr>
<th>Return result</th>
<th>Returns events for the specified door filtered by device, time interval and other optional parameters.</th>
</tr>
</thead>
</table>
  <events>  
    <event priority="2" evtTime="1237326065000" isAlarm="false" type="GW.EV_DOOR_FORCED_OPEN_CLEAR" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="168">  
      <description>Door Forced Open Cleared</description>  
    </event>  
    <event count="1" priority="8" evtTime="1237326060000" isAlarm="true" status="CLEARED" type="GW.EV_DOOR_FORCED_OPEN" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="167">  
      <description>Door Forced Open</description>  
    </event>  
    <event priority="2" evtTime="1237325666000" isAlarm="false" type="GW.EV_DOOR_FORCED_OPEN_CLEAR" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="160">  
      <description>Door Forced Open Cleared</description>  
    </event>  
    <event count="1" priority="8" evtTime="1237324372000" isAlarm="true" status="CLEARED" type="GW.EV_DOOR_FORCED_OPEN" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="159">  
      <description>Door Forced Open</description>  
    </event>  
    <event priority="1" evtTime="1237324363000" isAlarm="false" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="158">  
      <description>Door Opened</description>  
    </event>  
    <event priority="1" evtTime="1237316957000" isAlarm="false" deviceId="foGrYqQjT7yuxPDOw6/9Yg==" unid="157">  
      <description>Door Opened</description>  
    </event>  
  </events>  
</ns2:getAccessEventsByDoorResponse>` |

### getAllAlarmsByCategory

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllAlarmsByCategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lists alarms filtered by event category, time interval, and device ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
## Chapter 2  API Functions

### getAllAlarmsByType

**Function name**  
getAllAlarmsByType

**Description**  
Lists alarms filtered by event type, time interval, and device ID.

**Method**  
Post

**Parameters**

- `secCtx`—(required) the security context object returned by authenticateUser.
- `evntType`—(string: required) the event type.
- `beginTime`—(long: optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.
- `endTime`—(long: optional) the interval end time. Ending time for a time interval, specified as number of milliseconds from epoch time.
- `deviceId`—(optional) the unique ID of the device. If blank, all devices are included. If a device ID is included, events and alarms are only returned for the specified device.

### Sample request

The following XML example request specifies a category and time interval:

```xml
<ns2:getAllAlarmsByCategory xmlns:ns3="http://www.w3.org/2005/08/addressing" xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="7980237"/>
  <category>GW.EV_MAJOR_ID_DOOR</category>
  <beginTime>1244680318628</beginTime>
  <endTime>1247272318628</endTime>
</ns2:getAllAlarmsByCategory>
```

### Return result

Lists alarms filtered by event category, time interval, and device ID.

### Sample response

```xml
  <alarms>
    <event isAnnotation="false" siteId="LxQqj4cuTaO1YLINZk/dzA==" count="1" priority="8" evtTime="1247160358000" isAlarm="true" status="ACTIVE" type="GW.EV.DOOR_FORCED_OPEN" deviceId="szhUH331QKSq29LfkuVX/g==" unid="519">
      <description>Door Forced Open</description>
    </event>
  </alarms>
</ns2:getAllAlarmsByCategoryResponse>
```
### Sample request

The following XML example request specifies a category and time interval:

```xml
<ns2:getAllAlarmsByType xmlns:ns3="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://cisco.com/physec/acws/"
<secCtx token="2697030" />
<type>GW.EV_DOOR_FORCED_OPEN</type>
<beginTime>1244499763553</beginTime>
<endTime>1247091763553</endTime>
</ns2:getAllAlarmsByType>
```

### Return result

Returns the events and alarms by event type and other optional parameters.

### Sample response

```xml
<ns2:getAllAlarmsByTypeResponse
xmlns:ns2="http://cisco.com/physec/acws/
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<alarms>
<event evtTime="1228950257000"
isAlarm="true"
type="GW.EV_DOOR_FORCED_OPEN"
deviceId="GgXIKOh2Rciu33wuB5jXdA=="
unid="126">
<description>Door Forced Open</description>
</event>
</alarms>
</ns2:getAllAlarmsByTypeResponse>
```

---

**getAllAlarmUpdates**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllAlarmUpdates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Retrieves all alarm updates by getting a list of alarm annotations.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>

**Parameters**

- `secCtx`—(required) the security context object returned by `authenticateUser`.
- `beginTime`—(long:optional) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All events within this time interval are included in the result.
- `endTime`—(long: optional) the interval end time. Ending time for a time interval, specified as number of milliseconds from epoch time.

**Sample request**

```xml
<ns2:getAllAlarmUpdates xmlns:ns2="http://cisco.com/physec/acws/
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<secCtx token="32448635" />
<beginTime>1235259620358</beginTime>
<endTime>1237851620358</endTime>
</ns2:getAllAlarmUpdates>
```
### Return result

Returns a list of alarm annotation. Alarm annotation includes the following:

- **unid**—unique ID of annotation
- **alarmId**—alarm ID that it annotates.
- **state**—state of the related alarm.
- **siteId**—site identifier that owns the alarm annotation.
- **alarmDeviceId**—the unique ID of the device on which alarm is generated.

### Sample response

```xml
<ns2:getAllAlarmUpdatesResponse
xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
<annotations>
  <unid>11651</unid>
  <alarmId>11501</alarmId>
  <state>CLEARED</state>
  <siteId>RqNFHv3jQu6nwx9s9W4nkw==</siteId>
  <alarmDeviceId>hHHiu6oyQ8m6u+y/xP3j8A==</alarmDeviceId>
</annotations>
<annotations>
  <unid>11661</unid>
  <alarmId>11411</alarmId>
  <state>CLEARED</state>
  <siteId>RqNFHv3jQu6nwx9s9W4nkw==</siteId>
  <alarmDeviceId>hHHiu6oyQ8m6u+y/xP3j8A==</alarmDeviceId>
</annotations>
<annotations>
  <unid>11671</unid>
  <alarmId>811</alarmId>
  <state>CLEARED</state>
  <siteId>RqNFHv3jQu6nwx9s9W4nkw==</siteId>
  <alarmDeviceId>hHHiu6oyQ8m6u+y/xP3j8A==</alarmDeviceId>
</annotations>
<annotations>
  <unid>11681</unid>
  <alarmId>5631</alarmId>
  <state>CLEARED</state>
  <siteId>RqNFHv3jQu6nwx9s9W4nkw==</siteId>
  <alarmDeviceId>hHHiu6oyQ8m6u+y/xP3j8A==</alarmDeviceId>
</annotations>
<annotations>
  <unid>11781</unid>
  <alarmId>11741</alarmId>
  <state>CLEARED</state>
  <siteId>RqNFHv3jQu6nwx9s9W4nkw==</siteId>
  <alarmDeviceId>hHHiu6oyQ8m6u+y/xP3j8A==</alarmDeviceId>
</annotations>
</ns2:getAllAlarmUpdatesResponse></soap:Body></soap:Envelope>

---

**getAlarmInstructions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAlarmInstructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns information of the Alarm instruction.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters            | • secCtx: the security context object returned by authenticate User.  
                        • alarmId: specifies the unique alarm id. |
clearAlarm

<table>
<thead>
<tr>
<th>Function name</th>
<th>clearAlarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Clears a specified alarm.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—(required) the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>alarmId—(required) specifies the unique ID of the alarm</td>
</tr>
<tr>
<td></td>
<td>notes—(optional) notes that are saved with the alarm.</td>
</tr>
</tbody>
</table>

Sample request

```xml
<ns2:getAllAlarmUpdates xmlns:ns2="http://cisco.com/physec/acws/
 xmlns:ns3="http://www.w3.org/2005/08/addressing">
<secCtx token="32448635" />
<beginTime>1235259620358</beginTime>
<endTime>1237851620358</endTime>
</ns2:getAllAlarmUpdates>
```
Return result “True” if the request is successful.

Sample response

```xml
<ns2:getAllAlarmUpdatesResponse
  xmlns:ns2="http://cisco.com/physec/acws/"
  xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <annotations>
    <unid>165</unid>
    <alarmId>159</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>166</unid>
    <alarmId>123</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>169</unid>
    <alarmId>167</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>174</unid>
    <alarmId>167</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>175</unid>
    <alarmId>159</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>176</unid>
    <alarmId>123</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>245</unid>
    <alarmId>244</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  <annotations>
    <unid>254</unid>
    <alarmId>249</alarmId>
    <state>CLEARED</state>
    <siteId>AOL5p+ksQCKfBTBYQfumpQ==</siteId>
  </annotations>
  ...
</ns2:getAllAlarmUpdatesResponse>
```
### ackAlarm

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ackAlarm</td>
<td>Acknowledges an alarm.</td>
<td>Post</td>
<td>- secCtx—(required) the security context object returned by <code>authenticateUser</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- alarmId—(required) specifies the unique ID of the alarm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- notes—(string;optional) notes that are saved with the alarm that is acknowledged.</td>
</tr>
</tbody>
</table>

**Sample request**

```xml
<ns2:ackAlarm xmlns:ns2="http://cisco.com/physec/acws/"
               xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="25785386" />
  <eventId>382</eventId>
  <notes>This is alarm notes</notes>
</ns2:ackAlarm>
```

**Return result**

“True” if the request is successful.

**Sample response**

```xml
<ns3:ackAlarmResponse xmlns:ns2="http://www.w3.org/2005/08/addressing"
                       xmlns:ns3="http://cisco.com/physec/acws/">
  <result>true</result>
</ns3:ackAlarmResponse>
```

### getVersionInfo

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>getVersionInfo</td>
<td>Returns access control web service version information.</td>
<td>Post</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Sample request**

```xml
<ns2:getVersionInfo xmlns:ns2="http://cisco.com/physec/acws/"
                     xmlns:ns3="http://www.w3.org/2005/08/addressing">
</ns2:getVersionInfo>
```

**Return result**

The following information is returned:

- major version
- minor version
- release version
- build version

**Sample response**

```xml
                               xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <version build="265" maint="0" minor="1" major="1" />
</ns2:getVersionInfoResponse>
```
# getAlarmDuplicates

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAlarmDuplicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns duplicates for a specified alarm.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | • secCtx—(required) the security context object returned by authenticateUser.  
• eventId—(required) specifies the unique ID of the alarm. |
| Sample request| ```xml
<ns2:getAlarmDuplicates xmlns:ns2="http://cisco.com/physec/acws/
    xmlns:ns3="http://www.w3.org/2005/08/addressing">
    <secCtx token="25360171" />
    <alarmId>428451</alarmId>
</ns2:getAlarmDuplicates>
``` |
| Return result  | Returns a list of the duplicate alarms. |
| Sample response| ```xml
<ns2:getAlarmDuplicatesResponse
    xmlns:ns2="http://cisco.com/physec/acws/
    xmlns:ns3="http://www.w3.org/2005/08/addressing">
    <alarms>
    <event alarmDuplicateOf="428451" isAnnotation="false"
        siteId="d63KEPvzTBqujjBxWx1egQ==" count="2" priority="1"
        evtTime="1255022267000" isAlarm="true" status="CLEARED"
        type="GW.EV_REX_OPEN" deviceId="4uSOaOMRzKcOKqEw8TLpAw==" unid="428471">
        <description>REX Open</description>
        <srcDeviceName>rex</srcDeviceName>
    </event>
    </alarms>
</ns2:getAlarmDuplicatesResponse>
``` |
Event Notification

A client application can receive event and alarm notifications from Cisco PAM by registering a callback interface with the Cisco PAM web service.

To register a callback interface, do the following:

Step 1
Implement a service interface on the client.

Step 2
Register the service interface (endpoint) on Cisco PAM using either SOAP/HTTP or XML/HTTP.

- To implement a SOAP/HTTP binding on the client, use registerAcEventCallback. This is used to register a W3CEndPointReference for an endpoint service that implements AcEventPortType.
- To implement an XML/HTTP binding, use registerAcEventXMLCallback to register an endpoint URL.

Tip
Use unregisterAcEventCallback to unregister a callback endpoint.

Step 3
Use the notifyAcEvent API to notify a client application when an event or alarm occurs.

For more information, see the following:
- registerAcEventCallback, page 2-36
- registerAcEventXMLCallback, page 2-37
- unregisterAcEventCallback, page 2-38
- notifyAcEvent, page 2-39
- Example to Implement and Register a Service Interface, page 2-39

registerAcEventCallback

<table>
<thead>
<tr>
<th>Function name</th>
<th>registerAcEventCallback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Registers an event callback endpoint for clients using a SOAP/HTTP binding. When the specified events or alarms occur, a notification is sent to the registered endpoint.</td>
</tr>
<tr>
<td>Note</td>
<td>If an error is encountered in notifying events/alarms to the endpoint, the endpoint reference is automatically removed from the callback registry.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | • secCtx—(required) the security context object returned by authenticateUser.  
• evtTypes—(required) one or more event types for the notification.  
• evtCallback—(required) the callback interface used to notify event/alarms.  
• alarmsOnly—(required) true if only alarms are sent, False if events and alarm notifications are sent. |
Sample request

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns2="http://cisco.com/physec/acws/
   xmlns:ns3="http://www.w3.org/2005/08/addressing">
   <soap:Body>
     <ns2:registerAcEventCallback
         xmlns:ns2="http://cisco.com/physec/acws/
         xmlns:ns3="http://www.w3.org/2005/08/addressing">
       <secCtx token="8413945" />
       <eventCallback>
         <ns3:Address>http://localhost:8090/callback/notifyacevent</ns3:Address>
         <ns3:ReferenceParameters>
           <wsa:ReferenceParameters
               xmlns:wsa="http://www.w3.org/2005/08/addressing">
             <testParameter1>testValue1</testParameter1>
             <testParameter2>testValue2</testParameter2>
           </wsa:ReferenceParameters>
         </ns3:ReferenceParameters>
         <ns3:Metadata>
           <wsaw:ServiceName
               xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
               endpointname="AcEventPortTypePort">
             ns2:AcEventService</wsaw:ServiceName>
         </ns3:Metadata>
       </eventCallback>
       <alarmsOnly>false</alarmsOnly>
     </ns2:registerAcEventCallback>
   </soap:Body>
</soap:Envelope>
```

Return result

registrationKey—the key that identifies the callback registration. This value is returned when callback is registered successfully.
This key is also used to unregister the event callback. See unregisterAcEventCallback.

Sample response

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns2="http://cisco.com/physec/acws/
   xmlns:ns3="http://www.w3.org/2005/08/addressing">
   <soap:Body>
     <ns2:registerAcEventCallbackResponse
         xmlns:ns2="http://cisco.com/physec/acws/
         xmlns:ns3="http://www.w3.org/2005/08/addressing">
       <return>1001</return>
     </ns2:registerAcEventCallbackResponse>
   </soap:Body>
</soap:Envelope>
```

**registerAcEventXMLCallback**

<table>
<thead>
<tr>
<th>Function name</th>
<th>registerAcEventXMLCallback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Registers an event callback endpoint URL for clients using an XML/HTTP binding. When the specified events or alarms occur, a notification is sent to the registered URL. <strong>Note</strong> If a notification error occurs, the endpoint is automatically removed from the callback registry.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
### unregisterAcEventCallback

**Function name**  
unregisterAcEventCallback

**Description**  
Unregisters an event callback. This is called when application is no longer interested in getting notifications using the callback registered earlier.

**Method**  
Post

**Parameters**

- `secCtx`—(required) the security context object returned by `authenticateUser`.

- `registrationKey`—(required) the key that identifies the callback registration. This value is returned when callback is registered successfully. See `registerAcEventCallback`.

**Sample request**

```
<ns2:unregisterAcEventCallback xmlns:ns3="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="6846226" />
  <registrationKey>1007</registrationKey>
</ns2:unregisterAcEventCallback>
```

**Return result**  
None.

**Sample response**

```
<ns2:unregisterAcEventCallbackResponse xmlns:ns3="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://cisco.com/physec/acws/">
  <registrationKey>1007</registrationKey>
</ns2:unregisterAcEventCallbackResponse>
```
**notifyAcEvent**

<table>
<thead>
<tr>
<th>Function name</th>
<th>notifyAcEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Notifies a client application when an event or alarm occurs. The client must have previously registered a notification callback using either registerAcEventCallback or registerAcEventXMLCallback.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>AcEvent—event or alarm</td>
</tr>
</tbody>
</table>
| Sample request| ```
  <soap:Body>
    <ns2:notifyAcEvent>
      <event evtTime="1229014038427" isAlarm="true" type="GW.EV_DOOR_FORCED_OPEN" deviceId="GgXIKoh2Rciu33wuB5jXdA==" unid="168">
        <description>Door Forced Open</description>
      </event>
    </ns2:notifyAcEvent>
  </soap:Body>
</soap:Envelope>
``` |
| Return result | True if the event was handled. False if it was not. |
| Sample response| ```
  <soap:Body>
    <ns2:notifyAcEventResponse>
      <return>true</return>
    </ns2:notifyAcEventResponse>
  </soap:Body>
</soap:Envelope>
``` |

**Example to Implement and Register a Service Interface**

To receive event and alarm notifications, the client application must implement a service interface (AcEventPortType) defined in acws.wsdl as:

```java
public class AcEventPortTypeImpl implements AcEventPortType {

    public boolean notifyAcEvent(AcEvent event) {
        System.out.println("notifyAcEvent is called");
        // handle event/alarm as the case may be
        System.out.println(" Event = " + eventToString(event));
        return true;
    }
}
```

For example:

To create a service end point (for AcEventPortTypeImpl) and register it with PSIMWsPortType (ACWS web service), create an element `referenceParameters:`:

```xml
<wsa:ReferenceParameters/>
```

In the following XML fragment, the service endpoint is created and registered using the API registerAcEventCallback.

```xml
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsa:ReferenceProperties/>
</wsa:EndpointReference>
```
<wsa:ReferenceParameters>
    <testParameter1>testValue1</testParameter1>
    <testParameter2>testValue2</testParameter2>
</wsa:ReferenceParameters>

<wsa:Metadata
    xmlns:wsdli="http://www.w3.org/2006/01/wsd1-instance"
    xmlns:wsaw="http://www.w3.org/2005/02/addressing/wsdl"
    xmlns:ns="http://cisco.com/physec/acws/">
    <wsaw:ServiceName
        EndpointName="CallbackPort">ns:AcEventService
    </wsaw:ServiceName>
</wsa:Metadata>
</wsa:EndpointReference>

// code fragments for getting an endpoint reference
// and registering a callback.
    m_cbEndpointReference = (W3CEndpointReference)m_evtEndpoint.
                        getEndpointReference(referenceParameters);

    m_portType.registerAcEventCallback(m_ctx,
        null,
        m_cbEndpointReference,
        false);
Door Command APIs

grantDoorAccess

<table>
<thead>
<tr>
<th>Function name</th>
<th>grantDoorAccess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Grants access to a specified door using the Grant Door Access command.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters        | • secCtx—the security context object returned by authenticateUser.  
                   • doorId—door unique ID |
| Sample request    | `<ns2:grantDoorAccess xmlns:ns2="http://cisco.com/physec/acws/"  
                         xmlns:ns3="http://www.w3.org/2005/08/addressing">  
                         <secCtx token="27224694" />  
                         <doorId>fBU5hBu3RuegMSrmSfm5xg==</doorId>  
                         </ns2:grantDoorAccess>` |
| Return result     | True if the command is successful. False if it is not. |
| Sample response   | `<ns2:grantDoorAccessResponse xmlns:ns2="http://cisco.com/physec/acws/"  
                                xmlns:ns3="http://www.w3.org/2005/08/addressing">  
                                <result>true</result>  
                                </ns2:grantDoorAccessResponse>` |

setDoorMode

<table>
<thead>
<tr>
<th>Function name</th>
<th>setDoorMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Sets the door mode for a specified door and overrides the current door mode.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters        | • secCtx—the security context object returned by authenticateUser.  
                   • doorId—door unique ID.  
                   • mode—defines the new door mode (DoorModeType). |
| Sample request    | `<ns2:setDoorMode xmlns:ns2="http://cisco.com/physec/acws/"  
                    xmlns:ns3="http://www.w3.org/2005/08/addressing">  
                    <ctx token="24806017" />  
                    <doorId>PpDjVmWMSP69gjW/4nAvCg==</doorId>  
                    <mode>LOCK</mode>  
                    </ns2:setDoorMode>` |
| Return result     | None        |
| Sample response   | `<ns2:setDoorModeResponse xmlns:ns2="http://cisco.com/physec/acws/"  
                               xmlns:ns3="http://www.w3.org/2005/08/addressing" />` |

resetDoorMode

<table>
<thead>
<tr>
<th>Function name</th>
<th>resetDoorMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Resets the door mode to the original Cisco PAM configuration.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
### getDoorMode

**Function name**
getDoorMode

**Description**
Returns the current door mode for a specified door.

**Method**
Post

**Parameters**
- `secCtx`—the security context object returned by authenticateUser.
- `doorId`—door unique ID

**Sample request**
```xml
<ns2:getDoorMode xmlns:ns2="http://cisco.com/physec/acws/"
    xmlns:ns3="http://www.w3.org/2005/08/addressing">
    <secCtx token="13567485" />
    <doorId>PpDjVmWMSP69gjW/4nAvCg==</doorId>
</ns2:getDoorMode>
```

**Return result**
The door mode for the specified door.

**Sample response**
```xml
<ns2:getDoorModeResponse xmlns:ns2="http://cisco.com/physec/acws/"
    xmlns:ns3="http://www.w3.org/2005/08/addressing">
    <currentMode>OPEN</currentMode>
</ns2:getDoorModeResponse>
```

### setDoorModeConfig

**Function name**
setDoorModeConfig

**Description**
Sets the door mode configuration for a specified door. Any outstanding configuration changes from Cisco PAM are also applied.

**Method**
Post

**Sample request**
```xml
<ns2:setDoorModeConfig xmlns:ns2="http://cisco.com/physec/acws/"
    xmlns:ns3="http://www.w3.org/2005/08/addressing">
    <secCtx token="5428681" />
    <doorId>PpDjVmWMSP69gjW/4nAvCg==</doorId>
</ns2:setDoorModeConfig>
```

**Return result**
None.

**Sample response**
```xml
<ns2:setDoorModeConfigResponse xmlns:ns2="http://cisco.com/physec/acws/"
    xmlns:ns3="http://www.w3.org/2005/08/addressing" />
```
Parameters

- `secCtx`—the security context object returned by `authenticateUser`.
- `doorId`—unique ID of a door device in Cisco PAM. The door mode will be configured for the door specified in this parameter.
- `defMode`—the default door mode. The default mode can be one of the following:
  - OPEN
  - LOCK
  - SECURE
- `schName`—the name of the door schedule for changing door mode.
- `schMode`—the door mode used by the specified schedule.
  - OPEN
  - LOCK
  - SECURE

Sample request

```
<ns2:setDoorModeConfig xmlns:ns2="http://cisco.com/physec/acws/"
  xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <ctx token="31410451" />
  <doorId>PpDjVmWmSP69gjW4nAvCg==</doorId>
  <defMode>OPEN</defMode>
  <schName>Always Scheduled</schName>
  <schMode>LOCK</schMode>
</ns2:setDoorModeConfig>
```

Return result

None.

Sample response

```
<ns2:setDoorModeConfigResponse
  xmlns:ns2="http://cisco.com/physec/acws/"
  xmlns:ns3="http://www.w3.org/2005/08/addressing" />
```

---

**applyScheduleToDoor**

Function name: `applyScheduleToDoor`

Description: This API will apply the schedule to specified doors

Method: Post

Parameters

- `secCtx`—the security context object returned by `authenticateUser`.
- `scheduleUnId`—specifies the unique id of the schedule
- `doorUnId`—specifies the unique id of the door

Sample request

```
<ns2:applyScheduleToDoor xmlns:ns2="http://cisco.com/physec/acws/"
  xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="30563997" />
  <scheduleUnId>OhzbFdDBTOuLGtjp0mfr1w==</scheduleUnId>
  <doorUnId>EaOiklBuTZ6oV1RPu9cMGw==</doorUnId>
</ns2:applyScheduleToDoor>
```

Return result

Boolean flag indicating the success of the operation

Sample response

```
<ns2:applyScheduleToDoorResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <success>true</success>
</ns2:applyScheduleToDoorResponse>
```
### getAssociatedGWDoor

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAssociatedGWDoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns a list of associated devices [Door, GW] for the specified logical device Generic reader name in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | • secCtx—the security token returned by authenticate User.  
• Genericreader name—limits the results to a specific logical device Name. |
| Sample request| `<ns2:getAssociatedGWDoor xmlns:ns2="http://cisco.com/physec/acws/" xmlns:ns3="http://www.w3.org/2005/08/addressing">
<secCtx token="5121700"/>
<genreaderName>GR1</genreaderName>
</ns2:getAssociatedGWDoor>` |
| Return result | API returns the following values for the given device identifier:  
• IP Address  
• Port  
• Door Name |
<badge doorName="door1_codc" portNumber="4011" gwIpaddress="/10.78.177.232"/>
</ns2:getAssociatedGWDoorResponse>` |

### Badge Enrollment APIs

### getAllAccessLevels

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllAccessLevels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns detailed information for all access levels defined in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>• secCtx—the security context object returned by authenticateUser.</td>
</tr>
</tbody>
</table>
<ctx token="18492755" />
</ns2:getAllAccessLevels>` |
### getAllAccessLevelNames

**Function name** | getAllAccessLevelNames  
---|---
**Description** | Returns names of all the access level defined in Cisco PAM.  
**Method** | Post  
**Parameters** |  
  - secCtx—the security context object returned by authenticateUser.  
**Sample request** | <ns2:getAllAccessLevelNames xmlns:ns2="http://cisco.com/physec/acws/">
  <ctx token="27660152" />
  <alname>al1</alname>
</ns2:getAllAccessLevelNames>  
**Return result** | List of all access level names defined in Cisco PAM.  
**Sample response** | <ns2:getAllAccessLevelNamesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <accessLevelNames>al1</accessLevelNames>
  <accessLevelNames>lab access level</accessLevelNames>
  <accessLevelNames>al2</accessLevelNames>
  <accessLevelNames>Always Permit</accessLevelNames>
</ns2:getAllAccessLevelNamesResponse>

### getAccessLevelByByName

**Function name** | getAccessLevelByByName  
---|---
**Description** | Returns detailed information for a specified access level.  
**Method** | Post  
**Parameters** |  
  - secCtx—the security context object returned by authenticateUser.  
  - alname—name of access level.  
**Sample request** | <ns2:getAccessLevelByByName xmlns:ns2="http://cisco.com/physec/acws/">
  <ctx token="13605586" />
  <alname>al1</alname>
</ns2:getAccessLevelByByName>
**getAllScheduleNames**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllScheduleNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns names of all the access schedules defined in Cisco PAM.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>• secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>Sample request</td>
<td><code>&lt;ns2:getAllScheduleNames xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code>&lt;ctx token=&quot;171177710&quot; /&gt;<code>&lt;/ns2:getAllScheduleNames&gt;</code></td>
</tr>
<tr>
<td>Return result</td>
<td>List of all access schedules defined in Cisco PAM.</td>
</tr>
<tr>
<td>Sample response</td>
<td><code>&lt;ns2:getAllScheduleNamesResponse xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code>&lt;scheduleName&gt;Work Week Event Policy Schedule&lt;/scheduleName&gt;&lt;scheduleName&gt;Default Schedule&lt;/scheduleName&gt;&lt;scheduleName&gt;Always Permit Event Policy&lt;/scheduleName&gt;&lt;scheduleName&gt;Always Scheduled&lt;/scheduleName&gt;&lt;scheduleName&gt;Always Deny Event Policy&lt;/scheduleName&gt;&lt;scheduleName&gt;Work Schedule&lt;/scheduleName&gt;&lt;scheduleName&gt;Always Permit&lt;/scheduleName&gt;&lt;scheduleName&gt;Always Default&lt;/scheduleName&gt;<code>&lt;ns2:getAllScheduleNamesResponse&gt;</code></td>
</tr>
</tbody>
</table>
### getScheduleByName

**Function name**: getScheduleByName

**Description**: Returns the schedule name with corresponding value for a specified schedule.

**Method**: Post

**Parameters**:
- `secCtx`—the security context object returned by authenticate User
- `Name`—specifies the unique name of the schedule

**Sample request**:
```
<ns2:getScheduleByName xmlns:ns2="http://cisco.com/physec/acws/">
  <ctx token="8228712"/>
  <schName>Always Permit</schName>
</ns2:getScheduleByName>
```

**Return result**: Return the schedule name with corresponding values.

**Sample response**:
```
<ns2:getScheduleByNameResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <schedule scheduleType="ACCESSPOLICY" descr="Always Permit Schedule Description"
    name="Always Permit" unid="bCHPWybVQue1iMs2ZOPMpw==">
    <scheduleEntry order="0" scheduleAction="PERMIT">
      <ns2:scheduleElement>
        <timeEntryName>All Days Work Week</timeEntryName>
        <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
      </ns2:scheduleElement>
    </scheduleEntry>
  </schedule>
</ns2:getScheduleByNameResponse>
```

### enrollBadge

**Function name**: enrollBadge

**Description**: Adds one or more new badge credentials to an existing personnel record, or a new personnel record. If a new personnel record is specified, a picture in JPEG format can be included. Only new badges can be added using this API.

**Usage Notes**
A badge credential defines the properties for a user badge including one or more badge extensions. For example:
- The gwBadge extension defines properties such as credential template, use limit, and role.
- The HsBadge extension defines properties relevant for HSPD Badges.

**Note**
The Hs Badge extension is experimental and may be changed or removed in future Cisco PAM releases. For this reason, the extension should not be used in a production setting.

**Method**: Post
Parameters

- **secCtx**—the security context object returned by `authenticateUser`.
- **personInfo**—information for the personnel record for the assigned badge.
- **templateName**—the credential template name used to create the badge. Once the badge is created, a badge template specifies the initial properties of the badge.
- **BadgeList**—list of Badges where each of the badge can have one or more badge extensions specified.

Sample request

In the following sample:

- A `personId` identifies the person in the Cisco PAM. If the `personId` exists, the record is updated with the properties and badges information. If the `personId` does not exist in Cisco PAM, a new record is created for the `personId`.
- A Base64 encoded (JPEG) image is added to the record.
- A Badge Template (“TestTemplate”) is used to create the badges.

Sample request:

```xml
<ns2:enrollBadge xmlns:ns2="http://cisco.com/physsec/acws/">
  <secCtx token="22627865" />
  <person statusId="ACTIVE" type="Standard" lname="Last_5000" fname="First_5000" personId="247595849">
    <picture>
      /9j/4……………………..……………………..
      Note : Truncated base64 encoded JPEG Image
      ………………………………………
      ……………………………...
      ………………………………………2Q==
    </picture>
    <type>JPEG</type>
  </person>
  <templateName>TestTemplate</templateName>
  <badge>
    <badges cardNumber="1322936685508915001">
      <expDate>1252628974196</expDate>
      <validityType>VX.ACTIVE</validityType>
      <extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:HsBadge" ici="2" crlLatestDate="1252628974196" credentialSeries="1" credentialNumber="123456789ABCDEF0123456789ABCDEF0123456789ABCDFP0">
        <fascnHex>0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDFP0</fascnHex>
        <cuidHex>0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123</cuidHex>
        <sha1HashHex>0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123</sha1HashHex>
      </extensions>
      <extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:gwBadge">
        <credTemplateName>26BitWiegandCT</credTemplateName>
        <role>EMPLOYEE</role>
        <adaEnabled>true</adaEnabled>
      </extensions>
    </badges>
    <badges cardNumber="1322936685508915002">
      <expDate>1252628974196</expDate>
      <validityType>VX.ACTIVE</validityType>
    </badges>
  </badge>
</ns2:enrollBadge>
```
assignAccessLevels

Description
Assigns one or more access levels to a specified badge associated with a person.

Note
This API replaces all the access levels on the specified badge.

Method
Post

Parameters
- `secCtx`—the security context object returned by authenticateUser.
- `personUnid`—the unique ID of the personnel record.
- `badgeId`—the badge ID assigned to the specified person.
- `accessLevel`—a list of access level unique IDs assigned to the badge.

Sample request

```xml
<ns2:assignAccessLevel xmlns:ns2="http://cisco.com/physec/acws/"
xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="18492755" />
  <personUnid>D/DRqX4uToSMtLLXKO42LA==</personUnid>
  <badgeId>ObuzbsVOTQOECwkZ38v0uA==</badgeId>
  <accessLevel>iw3wxU4lSfSimYKGCFr+Ig==</accessLevel>
  <accessLevel>85MMlASIQFOxSkJeWrT6JA==</accessLevel>
</ns2:assignAccessLevel>
```

getPersonInfoByBadge

Description
Returns information for a personnel record for a given badge.

Method
Post

Parameters
- `secCtx`—the security context object returned by authenticateUser.
- `badge`—specifies the badge objects.
### getAllBadgesByPerson

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllBadgesByPerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns badge details for the badges assigned to a specified person.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>

**Parameters**
- `secCtx`—the security context object returned by `authenticateUser`.
- `personId`—the person unique ID.

**Sample request**
```xml
<ns2:getAllBadgesByPerson xmlns:ns2="http://cisco.com/physec/acws/
 xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <secCtx token="18492755"/>
  <personId>dte70tHfS2qSUEsGWADW/w==</personId>
</ns2:getAllBadgesByPerson>
```
**Chapter 2 API Functions**

### getBadgeValidityTypes

**Function name**
getBadgeValidityTypes

**Description**
Returns a list of Cisco PAM badge Validity types.

**Method**
Post

**Parameters**
- secCtx—the security context object returned by authenticateUser.

**Sample request**

```xml
<ns2:getBadgeValidityTypes
 xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="22627865" />
</ns2:getBadgeValidityTypes>
```

**Return result**
Returns a list of Cisco PAM badge Validity types. For example: Blacklist and Revoked.

**Tip**
To view the list in Cisco PAM, open the Badges module, click the General tab, and select the Validity drop-menu.

**Sample response**

```xml
<ns2:getBadgeValidityTypesResponse
 xmlns:ns2="http://cisco.com/physec/acws/">
  <types>GW.BLACKLIST</types>
  <types>HS.REVOKED</types>
  <types>VX.ACTIVE</types>
  <types>VX.DESTROYED</types>
  <types>VX.INACTIVE</types>
  <types>VX.LOST</types>
  <types>VX.STOLEN</types>
</ns2:getBadgeValidityTypesResponse>
```

### getCredentialTemplateNames

**Function name**
getCredentialTemplateNames

**Description**
Returns a list of names of Credential templates defined in CPAM

**Method**
Post

**Parameters**
- secCtx—the security context object returned by authenticateUser.

**Sample request**

```xml
<ns2:getCredentialTemplateNames
 xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="22627865" />
</ns2:getCredentialTemplateNames>
```

**Return result**
Returns a list of names of Credential templates defined in CPAM

**Sample response**

```xml
<ns2:getCredentialTemplateNamesResponse
 xmlns:ns2="http://cisco.com/physec/acws/">
</ns2:getCredentialTemplateNamesResponse>
```
## API Functions

### getBadgeRoles

**Function name**: getBadgeRoles  
**Description**: Returns the Badge type (roles) defined in Cisco PAM.  
**Method**: Post  
**Parameters**:  
- `secCtx`—the security context object returned by `authenticateUser`.  

**Sample request**

```xml
<ns2:getBadgeRoles xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="22627865" />  
</ns2:getBadgeRoles>
```

**Return result**: List of Badge type (roles) in Cisco PAM. For example: Employee, Contractor, Vendor, and Temporary.  

**Tip**: To view the list in Cisco PAM, open the Badges module, click the General tab, and select the Badge type drop-menu.  

**Sample response**

```xml
<ns2:getBadgeRolesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <roles>EMPLOYEE</roles>
  <roles>CONTRACTOR</roles>
  <roles>VENDOR</roles>
  <roles>TEMPORARY</roles>
</ns2:getBadgeRolesResponse>
```

### getAllBadgeTemplateNames

**Function name**: getAllBadgeTemplateNames  
**Description**: Returns the Badge Template names defined in Cisco PAM. The names are used as a parameter in the `enrollBadge` API.  
**Method**: Post  
**Parameters**:  
- `secCtx`—the security context object returned by `authenticateUser`.  

**Sample request**

```xml
<ns2:getCredentialTemplateNames xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="22627865" />  
</ns2:getCredentialTemplateNames>
```

**Return result**: List of Credential Template names. For example: 26BitWiegandCT and 26BitWiegandKeypadCT.  

**Tip**: To view the list in Cisco PAM, select Doors: Templates: Credential Templates.  

**Sample response**

```xml
<ns2:getCredentialTemplateNamesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <credTemplateNames>26BitWiegandCT</credTemplateNames>
  <credTemplateNames>26BitWiegandKeypadCT</credTemplateNames>
  <credTemplateNames>HID_CORP_1000_H500xx</credTemplateNames>
  <credTemplateNames>KeyPad_BCD4</credTemplateNames>
</ns2:getCredentialTemplateNamesResponse>
```
### getAccessLevelByPerson

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAccessLevelByPerson</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns a list of the access levels for all badges assigned to a person. The person is identified by a unique ID.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**         | • secCtx—the security context object returned by authenticateUser.  
• personUnid—unique ID that identifies the person. |
| **Sample request**     | `<ns2:getAccessLevelByPerson xmlns:ns2="http://cisco.com/physec/acws/"/>  
  <secCtx token="32189659" />  
  <personUnid>xcZJT7+QYKcjUzMmpC27w==</personUnid>` |
| **Return result**      | A list of the access levels for all badges held by the specified access control user. |
| **Sample response**    | `<ns2:getAccessLevelByPersonResponse xmlns:ns2="http://cisco.com/physec/acws/"/>  
  <accessLevels>LimitedAccess</accessLevels>` |

### getBadgeByCardNum

<table>
<thead>
<tr>
<th>Function name</th>
<th>getBadgeByCardNum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Retrieves a badge record for a specified card number.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**         | • secCtx—the security context object returned by authenticateUser.  
• cardNumber—the unique card number for the badge. |
| **Sample request**     | `<ns2:getBadgeByCardNum xmlns:ns2="http://cisco.com/physec/acws/"/>  
  <secCtx token="31616571" />  
  <cardNumber>1295</cardNumber>` |
| **Return result**      |                   |
| **Sample response**    |                   |
updateBadge

Function name | updateBadge
---|---
Description | Updates a list of badge credential in the Cisco PAM database. All properties on a badge in Cisco PAM are replaced by the new properties for the specified badge.
Method | Post
Parameters | - secCtx—the security context object returned by authenticateUser.
- BadgeList—the list of badges to update.
- personUnid—unique ID of person associated with the badges. If the specified ID does not exist, a fault is returned.
- assign—
  - enter true if the specified badges are associated with the specified person
  - enter false the association is removed.
Sample request | `<ns2:updateBadge xmlns:ns2="http://cisco.com/physec/acws/"`
  `<secCtx token="23402047" />
  `<badge>
    `<badges pin="5300" facilityCode="1" cardNumber="5300">
      `<badgeId id="213b20e+TtuxkGEvyBSIWQ="`
      `<expDate>1263715200000</expDate`
      `<type>VX.STANDARD</type>`
      `<site>qq09fJRiQqGmAPgaVkF+wQ==</site>`
      `<summaryData>5300</summaryData>`
      `<accessLevels>`
        `<accessLevels>`
      `<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:gwBadge">
        `<credTemplateName`26BitWiegandCT`</credTemplateName>
        `<role>EMPLOYEE</role>`
        `<execCredential>false</execCredential>`
        `<pinExempt>false</pinExempt>`
        `<adaEnabled>false</adaEnabled>`
      `<extensions>`
      `<extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:HsBadge">
        `<credId id="0" credentialSeries="0" credentialNumber="0" hsCardType="CARD_TYPE_NONE" agencyCode="0" />
      </extensions>`
    </badges>`
  </badge>`
  `<assign>true</assign>`
</ns2:updateBadge>`
<table>
<thead>
<tr>
<th>Return result</th>
<th>A fault is returned if no badge is found in the database.</th>
</tr>
</thead>
</table>
### getAllUpdatedBadges

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllUpdatedBadges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns all badges which are either modified or created between beginTime and now.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• secCtx—the security token returned by authenticate User.</td>
</tr>
<tr>
<td></td>
<td>• beginTime—(long) the interval start time. The beginning of a time interval, specified as number of milliseconds from epoch time. All Badges within this time interval are included in the result.</td>
</tr>
<tr>
<td>Sample request</td>
<td></td>
</tr>
</tbody>
</table>
|                     | <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
|                     |  <soap:Body>
|                     |   <ns2:getAllUpdatedBadges xmlns:ns2="http://cisco.com/physec/acws/">
|                     |    <secCtx token="19859305"/>
|                     |    <beginTime>1344557590611</beginTime>
|                     |   </ns2:getAllUpdatedBadges>
|                     |  </soap:Body>
|                     | </soap:Envelope> |
| Return result       | Detailed information of the badges. |
| Sample response     |                     |
|                     | <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
|                     |  <soap:Body>
|                     |   <ns2:getAllUpdatedBadgesResponse xmlns:ns2="http://cisco.com/physec/acws/">
|                     |    <badgeList>
|                     |     <badges facilityCode="0" cardNumber="522477" badgeId="d1lttjaERLeRJIPmmUL3WQ=="><type>VX.STANDARD</type><validityType>VX.ACTIVE</validityType><site>WBWCBNkJTF55n3t2oWz5kA==</site><summaryData>522477</summaryData><accessLevels>Aa7zXSxARFKtb2jbgnLRPQ==</accessLevels><extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:gwBadge"><credTemplateName>HID_CORP_1000_H500xx</credTemplateName><role>EMPLOYEE</role><execCredential>false</execCredential><pinExempt>false</pinExempt><adaEnabled>false</adaEnabled></extensions><extensions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:HsBadge" ici="0" credentialSeries="0" credentialNumber="0" hsCardType="CARD_TYPE_NONE" agencyCode="0"/></badges>
|                     | </badgeList>
|                     | </ns2:getAllUpdatedBadgesResponse>
|                     | </soap:Body>
|                     | </soap:Envelope> |

### getAllUpdatedPersonnels

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllUpdatedPersonnels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns details of all personnel [the first name, last name, user ID and unique ID] which are either modified or created between beginTime and now.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
getAccessLevelByUnid

Function name: getAccessLevelByUnid

Description: Returns information about an access level specified by the access level unid.

Method: Post

Parameters:
- **secCtx**—the security context object returned by authenticateUser.
- **unid**—the unique ID of an access level.

Sample request:

```xml
<ns2:getAccessLevelByUnid xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="21700915" />
  <unid>467cWZt+SK2EwMXXeTpyw==</unid>
</ns2:getAccessLevelByUnid>
```

Return result: Access level configuration based on the unid. If an invalid unid is specified, null is returned (the response is returned with no records).

Sample response:

```xml
<ns2:getAccessLevelByUnidResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <accessLevel>
    <ns2:accessLevelInfo unid="467cWZt+SK2EwMXXeTpyw==" deActivationDuration="0" enable="true" name="TestPolicy2">
      <ns2:accessLevelEntry doorId="rfghaHzARcu8E4Fhvv7hIw==" accesslevelentrytype="DOOR" scheduleName="Always Permit" />
    </ns2:accessLevelInfo>
  </accessLevel>
</ns2:getAccessLevelByUnidResponse>
```
## getAllBadgeTypes

**Function name**: getAllBadgeTypes

**Description**: Returns all the Badge types defined in Cisco PAM.

**Method**: Post

**Parameters**:
- secCtx—the security context object returned by authenticateUser.

**Sample request**:
```xml
<ns2:getAllBadgeTypeIds xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="23012302"/>
</ns2:getAllBadgeTypeIds>
```

**Return result**:
To view the list of the badge types in Cisco PAM:
1. In the Users list, click Badge.
2. Open a badge record, and click the General tab.
3. Click the Badge types menu to display the list.

**Sample response**:
```xml
<ns2:getAllBadgeTypeIdsResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <typeIds>SSS</typeIds>
  <typeIds>Standard</typeIds>
  <typeIds>Temporary</typeIds>
  <typeIds>Visitor</typeIds>
</ns2:getAllBadgeTypeIdsResponse>
```

## getAllBadgeRefs

**Function name**: getAllBadgeRefs

**Description**: Returns a range of badge references based on required beginning and ending card number parameters.

**Note**: A Badge Reference is an object that includes a badge unique ID and card number. The badge reference is used with the getBadgeByCardNum API to obtain complete badge information.

**Method**: Post
### Parameters

<table>
<thead>
<tr>
<th>Note</th>
<th>The beginCardNumber and endCardNumber parameters are required. The other parameters are optional and are used to further filter the results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>secCtx</td>
<td>the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>beginCardNumber</td>
<td>(long: required) Card numbers equal to or greater than this number are included in the results.</td>
</tr>
<tr>
<td>endCardNumber</td>
<td>(long: required) Card numbers equal to or less than this number are included in the results.</td>
</tr>
<tr>
<td>badgeTypeId</td>
<td>(string: optional) Only badges with the specified badge type are included in the results. If the specified badge type is invalid, a fault is returned.</td>
</tr>
<tr>
<td>validityTypeId</td>
<td>(string: optional) Only badges with the specified validity type are included in the results. If the specified validity type is invalid, a fault is returned.</td>
</tr>
<tr>
<td>onlyUnassigned</td>
<td>(boolean: required) if the value is true, only badges not associated to any person are included in the result. If the value is false, all badges are included.</td>
</tr>
</tbody>
</table>

### Sample request

```xml
<ns2:getAllBadgeRefs xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="11813864" />
  <beginCardNumber>5303</beginCardNumber>
  <endCardNumber>5307</endCardNumber>
  <onlyUnassigned>true</onlyUnassigned>
</ns2:getAllBadgeRefs>
```

### Return result

List of badges references, including the badge unique ID and card number for the reference.

### Sample response

```xml
<ns2:getAllBadgeRefsResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <badgeRef>
    <unid>vVPfk6p+Sc6rA/cVpWGmWg==</unid>
    <cardNumber>5305</cardNumber>
  </badgeRef>
</ns2:getAllBadgeRefsResponse>
```
# APIs for Time and Schedule based on Location

## createTimeEntry

<table>
<thead>
<tr>
<th>Function name</th>
<th>createTimeEntry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API creates a time entry record and returns a database unique ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>

**Parameters**

- `secCtx`—the security context object returned by authenticateUser.
- `timeEntry`—time entries could be recurring or non-recurring. Recurring types (Work Week, Special cases) contain rules (like recurring days of the week). Non-recurring types (Holiday aka date range) can be used to specify a date/date range. A time entry has a location attribute that assumes the location of the logged in user by default.

**Sample request**

```xml
<ns2:createTimeEntry xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="4607230"/>
  <timeEntry locationUnId="ohWZV6OSrmaKI1EtrxpBRA==" locationName="Campus">
    <holiday numberOfDays="0" name="CopyOf_Sam_Holiday">
      <startDate sec="1343890800000"/>
      <endDate sec="1343977200000"/>
    </holiday>
  </timeEntry>
</ns2:createTimeEntry>
```

**Return result**

Returns a time entry database unique ID

**Sample response**

```xml
<ns2:createTimeEntryResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <unid>+70GRyrY5WOD+tJQZrPSA==</unid>
</ns2:createTimeEntryResponse>
```

## updateTimeEntry

<table>
<thead>
<tr>
<th>Function name</th>
<th>updateTimeEntry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API updates the time entry record with the unique ID as specified in the timeEntry argument</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>

**Parameters**

- `secCtx`—the security context object returned by authenticateUser.
- `timeEntry`—specifies the time entry object.

**Sample request**

```xml
<ns2:updateTimeEntry xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="32263753"/>
  <timeEntry locationUnId="ohWZV6OSrmaKI1EtrxpBRA==" locationName="Campus" unid="y5yVYCWwSBcRmCj1s0Ejj==">
    <holiday numberOfDays="0" description="changed" name="sample_Holiday">
      <startDate sec="1344150000000"/>
      <endDate sec="1344236400000"/>
    </holiday>
  </timeEntry>
</ns2:updateTimeEntry>
```

**Return result**

No return result if the action is successful. A fault is returned if the time entry object is not found in the database. The following sample is for a successful action.

**Sample response**

```xml
<ns2:updateTimeEntryResponse xmlns:ns2="http://cisco.com/physec/acws/">
</ns2:updateTimeEntryResponse>
```
# deleteTimeEntry

<table>
<thead>
<tr>
<th>Function name</th>
<th>deleteTimeEntry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API deletes the time entry record referenced by the unique ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>~ secCtx—the security context object returned by authenticate User.</td>
<td></td>
</tr>
<tr>
<td>~ timeEntry—specifies the unique id of the time entry.</td>
<td></td>
</tr>
<tr>
<td>Sample request</td>
<td></td>
</tr>
</tbody>
</table>
| `<ns2:deleteTimeEntry xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="18940445"/>
  <unId>+70GRyrYS5WUm+tJQZZFSA==</unId>
</ns2:deleteTimeEntry>` |
| Return result       | Deleted time entry record with the uniqueld as specified in the timeEntry argument. Returns a Boolean flag indicating the success of the operation. |
| Sample response     |                 |
| `<ns2:deleteTimeEntryResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <success>true</success>
</ns2:deleteTimeEntryResponse>` |

# getAllTimeEntries

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllTimeEntries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API lists all the TimeEntries under the scope of the specified location. If the location is not specified, it will return all time entries in the logged in user's location scope (time entries in the sub-tree/ancestral locations).</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>~ secCtx—the security context object returned by authenticateUser</td>
<td></td>
</tr>
<tr>
<td>~ location—specifies the unique location at logged user's scope</td>
<td></td>
</tr>
<tr>
<td>Sample request</td>
<td></td>
</tr>
</tbody>
</table>
| `<ns2:getAllTimeEntries xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="18940445"/>
</ns2:getAllTimeEntries>` |
## API Functions

<table>
<thead>
<tr>
<th>Return result</th>
<th>Return the timentries under the scope of the login.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sample response</strong></td>
<td></td>
</tr>
</tbody>
</table>

```xml
<ns2:getAllTimeEntriesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <timeEntries unid="9uytwQqqQLODQi7rnTtLFw==">
    <workweek numberOfDays="7" description="All Days Work Week Schedule" name="All Days Work Week">
      <day>MONDAY</day>
      <day>SATURDAY</day>
      <day>FRIDAY</day>
      <day>TUESDAY</day>
      <day>WEDNESDAY</day>
      <day>SUNDAY</day>
      <day>THURSDAY</day>
    </workweek>
    <timeEntries unid="JhyJtcIeTY04hLg26QCeqQ==">
      <holiday numberOfDays="0" name="Sam_Holiday">
        <startDate sec="1343890800000"/>
        <endDate sec="1343977200000"/>
      </holiday>
    </timeEntries>
  </timeEntries>
</ns2:getAllTimeEntriesResponse>
```
### getTimeEntryById

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeEntryById</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the time entry id with corresponding value for a specified time entry.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters          | • secCtx—the security context object returned by authenticateUser.  
                     • idName—specifies the unique id of the time entry. |
| Sample request      | `<ns2:getTimeEntryById xmlns:ns2="http://cisco.com/physec/acws/">  
                       <secCtx token="26149632"/>  
                       <unId>9uytwQqqQLDQi7rnTtLFw==</unId>  
                       </ns2:getTimeEntryById>` |
| Return result       | Returns the time entry id with corresponding values. |
| Sample response     | `<soap:Body><ns2:getTimeEntryByIdResponse  
                       xmlns:ns2="http://cisco.com/physec/acws/">  
                       <timeEntry unid="9uytwQqqQLDQi7rnTtLFw==">  
                       <workweek numberOfDays="7" description="All Days Work Week Schedule" name="All Days Work Week">  
                       <day>MONDAY</day>  
                       <day>SATURDAY</day>  
                       <day>FRIDAY</day>  
                       <day>TUESDAY</day>  
                       <day>WEDNESDAY</day>  
                       <day>SUNDAY</day>  
                       <day>THURSDAY</day>  
                       </workweek>  
                       </timeEntry>  
                       </ns2:getTimeEntryByIdResponse>` |

### getTimeEntryByName

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeEntryByName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the time entry name with a corresponding value for a specified time entry.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters          | • secCtx—the security context object returned by authenticateUser.  
                     • name—specifies the unique name of the time entry. |
| Sample request      | `<ns2:getTimeEntryByName xmlns:ns2="http://cisco.com/physec/acws/">  
                       <secCtx token="11410489"/>  
                       <name>All Days Work Week</name>  
                       </ns2:getTimeEntryByName>` |
<table>
<thead>
<tr>
<th>Return result</th>
<th>Return the time entry name with corresponding values.</th>
</tr>
</thead>
</table>
| Sample response | `<ns2:getTimeEntryByNameResponse xmlns:ns2="http://cisco.com/physec/acws/">
<timeEntry unid="9uytwQqqQLODQi7rnTtLPw=="><workweek numberOfDays="7" description="All Days Work Week Schedule" name="All Days Work Week">
<day>MONDAY</day>
<day>SATURDAY</day>
<day>FRIDAY</day>
<day>TUESDAY</day>
<day>WEDNESDAY</day>
<day>SUNDAY</day>
<day>THURSDAY</day>
</workweek>
</timeEntry>
</ns2:getTimeEntryByNameResponse>` |
getAllSchedulesForTimeEntry

Function name | getAllSchedulesForTimeEntry
---|---
Description | Returns all schedules based on the specified time entry. This API supports time entries of the type Holiday and Workweek only.
Method | Post
Parameters | 
- `secCtx`—the security context object returned by authenticateUser.
- `timeEntryName`—specifies the unique name of the time entry.
Sample request | `<ns2:getAllSchedulesForTimeEntry xmlns:ns2="http://cisco.com/physec/acws/">  <secCtx token="32633843"/>  <timeEntryName>All Days Work Week</timeEntryName>  </ns2:getAllSchedulesForTimeEntry>`
Return result | Returns the time entry name: “All Days work Week” and associated schedules.
Sample response | `<ns2:getAllSchedulesForTimeEntryResponse xmlns:ns2="http://cisco.com/physec/acws/">  <schedules scheduleType="ACCESSPOLICY" descr="Always Permit Schedule Description" name="Always Permit" unid="TOvs2KUtSHOPscqOpzbtug==">  <scheduleEntry order="0" scheduleAction="PERMIT">  <ns2:scheduleElement>  <timeEntryName>All Days Work Week</timeEntryName>  <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>  </ns2:scheduleElement>  </scheduleEntry>  </schedules>  <schedules scheduleType="ACCESSPOLICY" descr="Always Deny Schedule Description" name="Always Deny" unid="k2SoG1yNqgiC6/tpO9kWag==">  <scheduleEntry order="0" scheduleAction="DENY">  <ns2:scheduleElement>  <timeEntryName>All Days Work Week</timeEntryName>  <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>  </ns2:scheduleElement>  </scheduleEntry>  </schedules>  ...` ```ns2:getAllSchedulesForTimeEntryResponse>

createTimeRangeGroup

Function name | createTimeRangeGroup
---|---
Description | This API creates a time range record and returns a database uniqueld.
Method | Post
Parameters | 
- `secCtx`—the security context object returned by authenticateUser.
- `timeRangeGroup`—As the name suggests, time range can be used to specify the exact time range. Time range in conjunction with time entry constitutes a schedule entry. A time range has a location attribute which assumes the location of the logged in user by default.
| Sample request | `<ns2:createTimeRangeGroup xmlns:ns2="http://cisco.com/physec/acws/">`<br>  
  
  `<secCtx token="8762090"/>`
  
  `<timeRangeGroup description="Always Permit/Deny Time Range Group" name="CopyOf_Always Time Range Group">`
  
  `<ns2:timeInterval end="86399999" begin="0"/>`
  
  `</timeRangeGroup>`
  
  `</ns2:createTimeRangeGroup>` |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return result</td>
<td>Return a time range group database unique ID.</td>
</tr>
</tbody>
</table>
| Sample response | `<ns2:createTimeRangeGroupResponse xmlns:ns2="http://cisco.com/physec/acws/">`<br>  
  
  `<unId>FT5X5I5ITLyq6Y9eNnoKUg==</unId>`
  
  `</ns2:createTimeRangeGroupResponse>` |
## updateTimeRangeGroup

<table>
<thead>
<tr>
<th>Function name</th>
<th>updateTimeRangeGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API updates the time range group record with the unique ID as specified in the time rangegroup argument.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters            | • secCtx—the security context object returned by authenticateUser.  
                       | • timeRangeGroup—specifies the time range group object. |
| Sample request        | <ns2:updateTimeRangeGroup xmlns:ns2="http://cisco.com/physec/acws/">  
                       |   <secCtx token="5838599"/>  
                       |   <timeRangeGroup unid="fL5LqZOXS469HsR7Ndf99A==" name="CopyOf_TimeRange"/>  
                       |   </timeRangeGroup>  
                       | </ns2:updateTimeRangeGroup> |
| Return result         | No return result if the action is successful. A fault is returned if the time range group object is not found in the database. |

## deleteTimeRangeGroup

<table>
<thead>
<tr>
<th>Function name</th>
<th>deleteTimeRangeGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API deletes the time range record identified by the unique ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters            | • secCtx—the security context object returned by authenticateUser.  
                       | • timeRangeGroup—specifies the unique id of the time range group. |
| Sample request        | <ns2:deleteTimeRangeGroup xmlns:ns2="http://cisco.com/physec/acws/">  
                       |   <secCtx token="10948521"/>  
                       |   <unId>DFPg2pERg1ALMNVJYKaZ4g==</unId>  
                       | </ns2:deleteTimeRangeGroup> |
| Return result         | Returns the deleted time range group record with the uniqueId as specified in the timeEntry argument. A boolean flag indicating the success of the operation. |
                       |   <success>true</success>  
                       | </ns2:deleteTimeRangeGroupResponse> |

## getAllTimeRangeGroups

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllTimeRangeGroups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API lists all the schedules under the scope of the specified location. If the location is not specified, it will return all schedules in the logged in user's location scope.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters            | • secCtx—the security context object returned by authenticateUser.  
                       | • location—specifies the unique location of the logged in user. |
| Sample request | `<ns2:getAllTimeRangeGroups xmlns:ns2="http://cisco.com/physec/acws/">
<secCtx token="8762090"/>
</ns2:getAllTimeRangeGroups>` |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return result</td>
<td>Return the schedules under the scope of the login user.</td>
</tr>
</tbody>
</table>
| Sample response| `<timeRangeGroups unid="I4r0z2cVQesK3NY8Wes1l7xQ==" description="Always Permit/Deny Time Range Group" name="Always Time Range Group">
<ns2:timeInterval end="86399999" begin="0"/>
</timeRangeGroups>` |
|                |                                                                                |
### getTimeRangeGroupById

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeRangeGroupById</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the time range group id with corresponding value for a specified time range group.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters             | • secCtx—the security context object returned by authenticate User.  
                          • id—specifies the unique id of the time range group. |
| Sample request         |  
                          <ns2:getTimeRangeGroupById xmlns:ns2="http://cisco.com/physec/acws/">  
                          <secCtx token="26813293"/>  
                          <unId>I4r0s2cVQeqK3KVrWe117xQ==</unId>  
                          </ns2:getTimeRangeGroupById> |
| Return result          | Return the time range group id with corresponding values. |
| Sample response        |  
                          <ns2:getTimeRangeGroupByIdResponse xmlns:ns2="http://cisco.com/physec/acws/">  
                          <TimeRangeGroup unid="I4r0s2cVQeqK3KVrWe117xQ==" description="Always Permit/Deny Time Range Group" name="Always Time Range Group">  
                          <ns2:timeInterval end="86399999" begin="0"/>  
                          </TimeRangeGroup>  
                          </ns2:getTimeRangeGroupByIdResponse> |

### getTimeRangeGroupByName

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeRangeGroupByName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the time range group name with corresponding value for a specified time range group.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters             | • secCtx—the security context object returned by authenticate User.  
                          • name—specifies the unique name of the time range group. |
| Sample request         |  
                          <ns2:getTimeRangeGroupByName xmlns:ns2="http://cisco.com/physec/acws/">  
                          <secCtx token="4035642"/>  
                          <name>Always Time Range Group</name>  
                          </ns2:getTimeRangeGroupByName> |
| Return result          | Returns the time range group name with corresponding values. |
| Sample response        |  
                          <ns2:getTimeRangeGroupByNameResponse xmlns:ns2="http://cisco.com/physec/acws/">  
                          <timeRangeGroup unid="I4r0s2cVQeqK3KVrWe117xQ==" description="Always Permit/Deny Time Range Group" name="Always Time Range Group">  
                          <ns2:timeInterval end="86399999" begin="0"/>  
                          </TimeRangeGroup>  
                          </ns2:getTimeRangeGroupByNameResponse> |

### createTimeEntryCollection

<table>
<thead>
<tr>
<th>Function name</th>
<th>createTimeEntryCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API creates a time entry collection as specified in the input argument and returns a unique database ID</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters             | • secCtx—the security context object returned by authenticateUser.  
                          • timeEntryCollection—(timeEntry and timeRange). |
Sample request

```xml
<ns2:createTimeEntryCollection xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="20623242"/>
  <timeEntryCollection locationUnId="ohWZV6OSRmaK1EtrxpBRA=="
    locationName="Campus" name="CopyOf_timeEntry_Collection">
    <ns2:timeEntryAndRange order="0">
      <timeEntryName>Sam_Holiday</timeEntryName>
      <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
    </ns2:timeEntryAndRange>
  </timeEntryCollection>
</ns2:createTimeEntryCollection>
```

Return result

Returns a Time Entry collection database unique ID.

Sample response

```xml
<ns2:createTimeEntryCollectionResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <unId>TZEIw6jQTDf+TF9pxqA==</unId>
</ns2:createTimeEntryCollectionResponse>
```
**updateTimeEntryCollection**

<table>
<thead>
<tr>
<th>Function name</th>
<th>updateTimeEntryCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This API updates the time entry collection record with the unique ID as specified in the time entry collection argument.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**        |   • secCtx—the security context object returned by authenticateUser.  
                                • timeEntryCollection—specifies the time entry collection objects. |

**Sample request**

```xml
<ns2:updateTimeEntryCollection xmlns:ns2="http://cisco.com/physec/acws/">
<secCtx token="29317576"/>
<timeEntryCollection locationUnId="ohWZv60SRmaKIlEtrxpBRA==" 
locationName="Campus" name="timeEntry_Collection" 
unid="M2KnZNKfS0h7T9LPAfgyuA==">
<ns2:timeEntryAndRange order="1"> 
<timeEntryName>Sam_Holiday</timeEntryName>
<timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
</ns2:timeEntryAndRange>
<ns2:timeEntryAndRange order="0"> 
<timeEntryName>CopyOf_Time</timeEntryName>
<timeRangeGroupName>Sample1</timeRangeGroupName>
</ns2:timeEntryAndRange>
</timeEntryCollection>
</ns2:updateTimeEntryCollection>
```

**Return result**

No return result if the action is successful. A fault is returned if the time entry collection object is not found in the database.

The following sample is for a successful action.

**Sample response**

```xml
</ns2:updateTimeEntryCollectionResponse>
```

**deleteTimeEntryCollection**

<table>
<thead>
<tr>
<th>Function name</th>
<th>deleteTimeEntryCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This API deletes the time entry collection record identified by the unique ID.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**        |   • secCtx—the security context object returned by authenticateUser.  
                                • timeEntryCollection—specifies the unique ID of the time entry collection. |

**Sample request**

```xml
<secCtx token="23539855"/>
<unId>TZEIwbJqTD+fY9pmqoA==</unId>
</ns2:deleteTimeEntryCollection>
```

**Return result**

Deleted time entry collection record with the uniqueld as specified in the timeEntryCollection argument. Returns a Boolean flag indicating the success of the operation.

**Sample response**

```xml
<ns2:deleteTimeEntryCollectionResponse xmlns:ns2="http://cisco.com/physec/acws/">
<success>true</success>
</ns2:deleteTimeEntryCollectionResponse>
```
### getAllTimeEntryCollections

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllTimeEntryCollections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This API lists all TimeEntryCollections under the scope of the specified location. If the location is not specified, it will return all time entry collections in the logged in user's location scope.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**    | • secCtx—the security context object returned by authenticateUser.  
                      • location—specifies the unique location of the logged in user. |
| **Sample request**| `<soap:Body><ns2:getAllTimeEntryCollections xmlns:ns2="http://cisco.com/physec/acws/">  
          <secCtx token="20623242"/>  
          </ns2:getAllTimeEntryCollections>` |
| **Return result** | Returns the Time Entry Collection under the scope of the login user. |
| **Sample response**| `<ns2:getAllTimeEntryCollectionsResponse xmlns:ns2="http://cisco.com/physec/acws/">  
          <TimeEntryCollections locationUnId="ohWZV608RmaKI1EtmgpBRA=="  
          locationName="Campus" name="timeEntry_Collection"  
          unid="M2KnZNKfS00t9lPAfgyuA==">  
          <ns2:timeEntryAndRange order="0">  
          <timeEntryName>Sam_Holiday</timeEntryName>  
          <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>  
          </ns2:timeEntryAndRange>  
          </TimeEntryCollections>  
          .........  
          .........  
          </ns2:getAllTimeEntryCollectionsResponse>` |

### getTimeEntryCollectionById

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeEntryCollectionById</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Returns the time entry collection id with corresponding value for a specified time entry collection.</td>
</tr>
<tr>
<td><strong>Method</strong></td>
<td>Post</td>
</tr>
</tbody>
</table>
| **Parameters**    | • secCtx—the security context object returned by authenticateUser.  
                      • id—specifies the unique id of the time entry collection. |
| **Sample request**| `<ns2:getTimeEntryCollectionById xmlns:ns2="http://cisco.com/physec/acws/">  
          <secCtx token="66923"/>  
          <unId>9uIUZcaXqgezBNLH0lmuQ==</unId>  
          </ns2:getTimeEntryCollectionById>` |
| **Return result** | Returns the time entry collection id with corresponding values. |
| **Sample response**| `<ns2:getTimeEntryCollectionByIdResponse xmlns:ns2="http://cisco.com/physec/acws/">  
          <timeEntryCollection name="TEC" unid="9uIUZcaXqgezBNLH0lmuQ==">  
          <ns2:timeEntryAndRange order="0">  
          <timeEntryName>sample_Holiday</timeEntryName>  
          <timeRangeGroupName>sample_Time</timeRangeGroupName>  
          </ns2:timeEntryAndRange>  
          </timeEntryCollection>  
          </ns2:getTimeEntryCollectionByIdResponse>` |
### getTimeEntryCollectionByName

<table>
<thead>
<tr>
<th>Function name</th>
<th>getTimeEntryCollectionByName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns the time entry collection name with corresponding value for a specified time entry collection.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>• Name—specifies the unique name of the time entry collection.</td>
</tr>
<tr>
<td>Sample request</td>
<td><code>&lt;ns2:getTimeEntryCollectionByName xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;secCtx token=&quot;5416654&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;name&gt;TEC&lt;/name&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getTimeEntryCollectionByName&gt;</code></td>
</tr>
<tr>
<td>Return result</td>
<td>Returns the time entry collection name with corresponding values.</td>
</tr>
<tr>
<td>Sample response</td>
<td><code>ns2:getTimeEntryCollectionByNameResponse xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryCollection name=&quot;TEC&quot; unid=&quot;9uIUXcaXqezBNLHO1omuQ==&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryAndRange order=&quot;0&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryName&gt;sample_Holiday&lt;/timeEntryName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeRangeGroupName&gt;sample_Time&lt;/timeRangeGroupName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/timeEntryCollection&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getTimeEntryCollectionByNameResponse&gt;</code></td>
</tr>
</tbody>
</table>

### getAllSchedulesForTimeEntryCollection

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllSchedulesForTimeEntryCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Returns all Schedules based on the specified time entry collection.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>• timeEntryCollectionName—specifies the unique name of the time entry collection</td>
</tr>
<tr>
<td>Sample request</td>
<td><code>&lt;ns2:getAllSchedulesForTimeEntryCollection xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;secCtx token=&quot;965546&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryCollectionName&gt;MM&lt;/timeEntryCollectionName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getAllSchedulesForTimeEntryCollection&gt;</code></td>
</tr>
</tbody>
</table>
### Note

Time entries, Time ranges and Time entry collections can be shared between multiple schedule entries and any changes to them are reflected in all associated schedules.

---

| Return result | Returns the time entry collection name: “MM” associated schedules. |

**Sample response**

```xml
<ns2:getAllSchedulesForTimeEntryCollectionResponse
  xmlns:ns2="http://cisco.com/physsec/acws/">
  <schedules locationUnId="5i1rJX2oRW+7Ejcs9E7fRA==" locationName="HCL"
    scheduleType="ACCESSPOLICY" name="S_AP" unid="JdkNsN8XTRi+4IFMoCJh1g=="/>
    <scheduleEntry order="0" scheduleAction="DENY">
        <timeEntryCollection locationUnId="5i1rJX2oRW+7Ejcs9E7fRA==" locationName="HCL"
            name="MM" unid="uJA0amfvQrmuOtXH7AyWKA==">
                <ns2:timeEntryAndRange order="0">
                    <timeEntryName>sam</timeEntryName>
                    <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
                </ns2:timeEntryAndRange>
            </timeEntryCollection>
        </scheduleEntry>
    </schedules>
    <schedules locationUnId="5i1rJX2oRW+7Ejcs9E7fRA==" locationName="HCL"
      scheduleType="DOORPOLICY" name="S_DP" unid="5ScNKd3ZR56ruK0Q68AxXQ=="/>
    <scheduleEntry order="0" scheduleAction="PERMIT">
      <timeEntryCollection locationUnId="5i1rJX2oRW+7Ejcs9E7fRA==" locationName="HCL"
        name="MM" unid="uJA0amfvQrmuOtXH7AyWKA==">
          <ns2:timeEntryAndRange order="0">
            <timeEntryName>sam</timeEntryName>
            <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
          </ns2:timeEntryAndRange>
        </timeEntryCollection>
    </scheduleEntry>
  </schedules>
  ....
  ....
</ns2:getAllSchedulesForTimeEntryCollectionResponse>
```
### createSchedule

<table>
<thead>
<tr>
<th>Function name</th>
<th>createSchedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API creates a schedule as specified in the input argument and returns a unique database ID.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | secCtx—the security context object returned by authenticateUser.  
|               | schedule—(The time entries, time ranges and time entry collections). |

**Sample request**

```xml
<ns2:createSchedule xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="15441492"/>
  <schedule locationUnId="ohWZV60SKmaK1lEtrxpBRA==" locationName="Campus" scheduleType="DOORPOLICY" name="CopyOf_Sample_DP">
    <scheduleEntry order="0" scheduleAction="PERMIT">
      <ns2:scheduleElement><timeEntryName>Sam_Holiday</timeEntryName>
      <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
    </ns2:scheduleElement>
  </scheduleEntry>
</schedule>
</ns2:createSchedule>
```

**Return result**

Returns a Schedule database unique ID.

**Sample response**

```xml
<ns2:createScheduleResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <unid>3IwJzrutRoODf/OrvkyJUw==</unid>
</ns2:createScheduleResponse>
```

### updateSchedule

<table>
<thead>
<tr>
<th>Function name</th>
<th>updateSchedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API updates the schedule record with the unique ID as specified in the schedule argument.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | secCtx—the security context object returned by authenticate User.  
|               | schedule—specifies the schedule objects. |

**Sample response**

```xml
  <unid>3IwJzrutRoODf/OrvkyJUw==</unid>
</ns2:updateScheduleResponse>
```
| Sample request | `<ns2:updateSchedule xmlns:ns2="http://cisco.com/physec/acws/">`<secCtx token="25703729="/>
          <schedule scheduleType="ACCESSPOLICY" descr="All working days 8AM to 6PM schedule " name="Default Schedule" unid="T2/FBztoQwC2jRnlrfzw6g=="><scheduleEntry order="1" scheduleAction="PERMIT"><ns2:scheduleElement><timeEntryName>sample_Holiday</timeEntryName><scheduleEntry></scheduleEntry><scheduleEntry order="2" scheduleAction="PERMIT"><ns2:scheduleElement><timeEntryName>sample_Holiday</timeEntryName><timeRangeGroupName>copyOf_sample_Time</timeRangeGroupName></ns2:scheduleElement></scheduleEntry><scheduleEntry order="0" scheduleAction="PERMIT"><ns2:scheduleElement><timeEntryName>WWAdminTest</timeEntryName><timeRangeGroupName>TR Admin - always</timeRangeGroupName></ns2:scheduleElement></scheduleEntry></schedule></ns2:updateSchedule>` |
| Return result | No return result if the action is successful. A fault is returned if the schedule object is not found in the database. The following sample is for a successful action. |
### addExceptionToSchedule

**Function name**  
addExceptionToSchedule

**Description**  
This API adds the provided schedule entry as an exception in the specified schedule. This method adds the given schedule entry to the schedule at the order specified in the schedule entry.

**Method**  
Post

**Parameters**  
- `secCtx`—the security token returned by authenticate User.
- `ScheduleName`—specifies the unique name of the schedule.
- `ScheduleEntry`—(time Entry and time Range or time entry collection).

**Sample request**  
```xml  
<ns2:addExceptionToSchedule xmlns:ns2="http://cisco.com/physec/acws/"><secCtx token="29063449"/>
   <scheduleName>sample</scheduleName>
   <scheduleEntry scheduleAction="PERMIT">
      <timeEntryCollection locationUnId="5i1rJX2oRW+7Ejcs9E7fRA==" locationName="HCL" name="TEC1" unid="5ysW4Sm0TPeQ5HCvEfeLGw=">
         <ns2:timeEntryAndRange order="0"><timeEntryName>sam</timeEntryName>
            <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
         </ns2:timeEntryAndRange>
      </timeEntryCollection>
   </scheduleEntry>
</ns2:addExceptionToSchedule>  
```

**Return result**  
Any schedule entries that exist in the given order will be moved down to accommodate this schedule entry. Returntype is void.

**Sample response**  
```xml  
<ns2:addExceptionToScheduleResponse xmlns:ns2="http://cisco.com/physec/acws/">
   <success>true</success>
</ns2:addExceptionToScheduleResponse>  
```

### deleteSchedule

**Function name**  
deleteSchedule

**Description**  
This API deletes the schedule record identified by the unique ID of the input argument along with the schedule entry (ies).

**Method**  
Post

**Parameters**  
- `secCtx`—the security context object returned by authenticateUser.
- `schedule`—specifies the unique id of the schedule.

**Sample request**  
```xml  
<ns2:deleteSchedule xmlns:ns2="http://cisco.com/physec/acws/">
   <secCtx token="12457024"/>
   <unid>3IwJzrutRoODf/OrvkyJUw==</unid>
</ns2:deleteSchedule>  
```

**Return result**  
Returns a Boolean flag indicating the success of the operation.

**Sample response**  
```xml  
<ns2:deleteScheduleResponse xmlns:ns2="http://cisco.com/physec/acws/">
   <success>true</success>
</ns2:deleteScheduleResponse>  
```
### getAllSchedules

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllSchedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API lists all the schedules under the scope of the specified location. If the location is not specified, it will return all schedules in the logged in user’s location scope.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>location—specifies the unique location at logged user’s scope.</td>
</tr>
<tr>
<td>Sample request</td>
<td><code>&lt;ns2:getAllSchedules xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;ctx token=&quot;8687673&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;locationUnId&gt;ohWZV6OSRmaKi1EtrxpBRA==&lt;/locationUnId&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getAllSchedules&gt;</code></td>
</tr>
<tr>
<td>Return result</td>
<td>Return the schedules under the scope of the specified location.</td>
</tr>
<tr>
<td>Sample response</td>
<td><code>&lt;ns2:getAllSchedulesResponse xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;schedule locationUnId=&quot;ZG5WEAO9Tkac3crQvBo26w==&quot; locationName=&quot;Building1&quot; scheduleType=&quot;ACCESSPOLICY&quot; name=&quot;Sam_AP&quot; unid=&quot;xT/LIB+nSUKPw9rqY0pIcg==&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;scheduleEntry order=&quot;0&quot; scheduleAction=&quot;DENY&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;ns2:scheduleElement&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryName&gt;Sam_Holiday&lt;/timeEntryName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeRangeGroupName&gt;Always Time Range Group&lt;/timeRangeGroupName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:scheduleElement&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/scheduleEntry&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/schedule&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>...........</code></td>
</tr>
<tr>
<td></td>
<td><code>...........</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;schedule locationUnId=&quot;ohWZV6OSRmaKi1EtrxpBRA==&quot; locationName=&quot;Campus&quot; scheduleType=&quot;DOORPOLICY&quot; name=&quot;Sample_DP&quot; unid=&quot;OhzbFdDBTOuLGtjp0mfr1w==&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;scheduleEntry order=&quot;0&quot; scheduleAction=&quot;PERMIT&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;ns2:scheduleElement&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeEntryName&gt;Sam_Holiday&lt;/timeEntryName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timeRangeGroupName&gt;Always Time Range Group&lt;/timeRangeGroupName&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:scheduleElement&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/scheduleEntry&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/schedule&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getAllSchedulesResponse&gt;</code></td>
</tr>
</tbody>
</table>

### getAllDoorSchedules

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllDoorSchedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API lists all the door schedules under the scope of the specified location. If the location is not specified, it will return all schedules in the logged in user’s location scope.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td></td>
<td>location—specifies the unique location of the logged in user.</td>
</tr>
<tr>
<td>Sample request</td>
<td><code>&lt;ns2:getAllDoorSchedules xmlns:ns2=&quot;http://cisco.com/physec/acws/&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;secCtx token=&quot;877261&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/ns2:getAllDoorSchedules&gt;</code></td>
</tr>
</tbody>
</table>
### Return result

Return all the door schedules under the scope of the specified location.

### Sample response

```xml
<ns2:getAllDoorSchedulesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <schedules locationUnId="ohWZV6OSRmaKIlEtrxpBRA==" locationName="Campus" scheduleType="DOORPOLICY" name="Sample_DP" unid="OhzbFdDBTOuLjtj0mfr1w==">
    <scheduleEntry order="0" scheduleAction="PERMIT">
      <ns2:scheduleElement>
        <timeEntryName>Sam_Holiday</timeEntryName>
        <timeRangeGroupName>Always Time Range Group</timeRangeGroupName>
      </ns2:scheduleElement>
    </scheduleEntry>
  </schedules>
......
......
</ns2:getAllDoorSchedulesResponse>
```

### Note

- For the "Use Scheduled Mode" action type use
  
  ```java
  scheduleEntry.setScheduleAction(AccessControlActionValue.PERMIT)
  ```

- For the "Use Default Mode" action type use
  
  ```java
  scheduleEntry.setScheduleAction(AccessControlActionValue.DENY)
  ```
### getAllSchedulesForTimeRangeGroup

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllSchedulesForTimeRangeGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API returns the specify time range group name argument associated with schedules.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
</tbody>
</table>
| Parameters    | - `secCtx`—the security context object returned by authenticate User.  
                - `name`—specifies the unique name of the time range group. |
| Sample request| `<ns2:getAllSchedulesForTimeRangeGroup xmlns:ns2="http://cisco.com/physec/acws/"><secCtx token="14254389"/></ns2:getAllSchedulesForTimeRangeGroup>`  |
| Return result | Returns the schedule associated with time range group value “Always Time Range Group”. |
| Sample response| `<ns2:getAllSchedulesForTimeRangeGroupResponse xmlns:ns2="http://cisco.com/physec/acws/">  
<schedules scheduleType="ACCESSPOLICY" descr="Always Permit Schedule Description" name="Always Permit" unid="bCHPWybVQueliMs2ZOPMpwm="">  
<scheduleEntry order="0" scheduleAction="PERMIT">  
<ns2:scheduleElement>  
<timeEntryName>All Days Work Week</timeEntryName>  
<timeRangeGroupName>Always Time Range Group</timeRangeGroupName>  
</ns2:scheduleElement>  
</scheduleEntry>  
</schedules>` |

**Note**: A Schedule Entry can only be associated with one schedule and cannot be shared between schedules.
APIs for Location Listing

**getAllLocationsInScope**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAllLocationsInScope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API provides a list of locations in the sub-tree of the logged-in user’s default location.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—the security context object returned by authenticateUser.</td>
</tr>
<tr>
<td>Sample request</td>
<td></td>
</tr>
<tr>
<td>Return result</td>
<td>Return the locations at the login user’s scope.</td>
</tr>
<tr>
<td>Sample response</td>
<td></td>
</tr>
</tbody>
</table>

**getAncestorLocations**

<table>
<thead>
<tr>
<th>Function name</th>
<th>getAncestorLocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This API returns all the locations leading to the logged in user’s location starting from the root of the location hierarchy.</td>
</tr>
<tr>
<td>Method</td>
<td>Post</td>
</tr>
<tr>
<td>Parameters</td>
<td>secCtx—the security context object returned by authenticate User.</td>
</tr>
<tr>
<td>Sample request</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Return result

Returns all the locations leading to the logged in user’s location starting from the root of the location hierarchy.

### Sample response

```xml
<ns2:getAncestorLocationsResponse xmlns:ns2="http://cisco.com/physec/acws/
xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <retList name="Base" unid="ohWZWV60SExmaKI1EltrxpBRA==">
    <children name="Base/Building1" unid="ZG5WEOA09Tkac3crQvBo26w==">
      <children name="Base/Building1/Floor1" unid="YdaI2zsS1+cEmuAH73pEA=="/>
    </children>
    <children name="Base/Campus" unid="yUauEDKZR26tE3etZj7KBw==">
      <children name="Base/Campus/Building2" unid="yIk7D0PHtmtqF91eq1bXbA==">
        <children name="Base/Campus/Building2/Floor1" unid="IhA647mSRXX9ETD6xqO51A=="/>
      </children>
    </children>
  </retList>
  <retList name="Base/Campus" unid="yUauEDKZR26tE3etZj7KBw==">
    <children name="Base/Campus/Building2" unid="yIk7D0PHtmtqF91eq1bXbA==">
      <children name="Base/Campus/Building2/Floor1" unid="IhA647mSRXX9ETD6xqO51A=="/>
    </children>
  </retList>
</ns2:getAncestorLocationsResponse>
```
## API for Configuration changes

### applyConfigurationChanges

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyConfigurationChanges</td>
<td>Once all the related tasks in the configuration sequence are applied, this API can be used for pushing the changes to the gateways in the logged in user's location scope.</td>
<td>Post</td>
<td>• secCtx—the security context object returned by authenticate User.</td>
</tr>
</tbody>
</table>

#### Sample request
```
<ns2:applyConfigurationChanges xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="9892076"/>
  <gateways enabled="true" siteId="JEdf71O1TniD3mF/k9U6AA=" deviceState="GW.UP" address="/gw_FOC144888H4" type="GW.CTLR" name="gw_FOC144888H4" unid="Mu72HyLKhG6d19Ho1vQA="/>
</ns2:applyConfigurationChanges>
```

#### Return result
Returns a Boolean flag indicating the success of the operation.

#### Sample response
```
<ns2:applyConfigurationChangesResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <success>true</success>
</ns2:applyConfigurationChangesResponse>
```

### applyConfigurationChangesToLocation

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| applyConfigurationChangesToLocation | Pushes the configuration changes to gateways in the specified location. | Post   | • secCtx—the security context object returned by authenticate User.  
  • locationUnId—specifies the unique location id. |

#### Sample request
```
<ns2:applyConfigurationChangesToLocation xmlns:ns2="http://cisco.com/physec/acws/">
  <secCtx token="9989478"/>
  <locationUnId>1sRrEdLcRZ2yYVAFjdIrfA==</locationUnId>
</ns2:applyConfigurationChangesToLocation>
```

#### Return result
Returns a Boolean flag indicating the success of the operation.

#### Sample response
```
<ns2:applyConfigurationChangesToLocationResponse xmlns:ns2="http://cisco.com/physec/acws/">
  <success>true</success>
</ns2:applyConfigurationChangesToLocationResponse>
```

## Additional Information

### Precursor to APIs based on Location

- Create a location hierarchy.
- Create Profiles on the CPAM client for the different roles.
Additional Information

- Create Login users to assign the created profiles along with the hierarchial locations specific to each profile.
- Create default schedules for the doors and alarms in different areas within the hierarchial location.
- Set the location of the schedules as well as other time elements so that they are accessible to the login users.

Sample Usecases

To enable users to set a default schedule for each day in a week

- Identify the default recurring time intervals that would apply to a set of buildings in a campus.
- Create schedules with these time intervals. Example: Work week time entry – M-F; Weekend time entry – Sat, Sun; University holidays; Time range 8AM-5PM

EXCEPTIONS

- Recurring – recurring exceptions should replace the default time entries for that location.
- Non-recurring – Non-recurring exceptions like specific dates could be added apart from the default time entries and take precedence.

Note: Ensure to have one recurring work week type of time entry as the default in a schedule and add exceptions for specific dates apart from this default entry.

APIS TO USE

- createTimeEntry, page 2-60 (create one for each day of the week)
- createSchedule, page 2-75

Note: Use the time entries created above with a time range group (use getTimeEntryByName, getTimeRangeGroupByName)

To enable users to add specific Time Changes one day at a time

APIS TO USE

- getAllTimeEntries
- getTimeEntryByName
- getAllTimeRangeGroups
- getTimeRangeGroupByName

To enable users to modify the Time element

Use the following APIs to update all the schedules/time entry collections that use a time element

- updateTimeEntry
- updateTimeRangeGroup
To enable an user to modify the default schedule on any day or all days

Consider the following example:

A building changes its operating hours from 8am-5pm to 7am-6pm M-F.

Use the following API to change the default schedule on each day to the new 7am-6pm schedule.

- getTimeRangeGroupByName(TimeRangeGroup_8-5)

To modify the time range group, use the following API:

- updateTimeRangeGroup

The updateTimeRangeGroup API internally updates all the schedules/time entry collections that use the time range group.

Note: Ensure that the existing time range is already assigned to a schedule in conjunction with a time entry.

Example: Monday_Schedule has WorkWeek_M-F and TimeRangeGroup_8-5

To enable users to modify the schedule on one particular day

Consider the following example:

A holiday is coming up. If the user wishes to modify the schedule on one particular day (without disturbing the default schedule) so that the door(s)/area(s) they administer would be locked down for the specific holiday. If the holiday consists of more than one day then the user has to set the holiday schedule for each day, independent of the others.

Use the following API to create a time entry collection specifying all the holidays.

- createTimeEntryCollection
- addExceptionToSchedule

The addExceptionToSchedule API adds an exception to a schedule specifying a holiday or a time entry collection.

To enable users to change Alarm schedules for Buildings

By changing the time range all the associated door schedules as well as the alarm schedules are internally updated. The precursor to this use case is that at the time of initial configuration, two sets of schedules are created (door schedules as well as event policy type schedules). The alarm schedules should be used in Global IO rules that are created for every building to notify the local security guard in case of a door jam/door forced open etc.

Note: The event policy schedules can be created from the CPAM client only.

Recording External Events

External applications can record events in Cisco PAM using the recordExtEvent API. Once recorded, the events are displayed in the Events & Alarms Monitoring modules.

External Event Types are defined using the Event Definition Format and imported using the steps described in the following sections.
To record events from external applications, do the following:

1. **Define External Event Types Using the Event Definition Format**, page 2-86. This file also defines the categories for the log codes.
2. **Create a Text File to Define the Event Names in Cisco PAM**, page 2-86.
3. **Import the Files into Cisco PAM**, page 2-87.
4. Add external events and alarms to Cisco PAM using recordExtEvent, page 2-89. See also the **Examples: Using Java and C# Code**, page 2-89.

**Define External Event Types Using the Event Definition Format**

Use the Event Definition Format to create an XML file that defines the event and alarm codes used to add external events to Cisco PAM. This file also defines the category for the events and is imported into Cisco PAM to create the codes.

**Example**

In the following XML example:

- The concatenation rule is: AE.<logcode_prefix>_<logcode>
- Event category: AE.Cisco_VSM
- The log codes for the category are: AE.VS_VSM_Sample1 and AE.VS_VSM_Sample2

```
<appext_eventdefns
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <appext_entry appname="Cisco_VSM" logcode_prefix="VS">
  <ext_event_defn logcode="VSM_Sample1" priority="10" description="VSM Sample Event-1"/>
  <ext_event_defn logcode="VSM_Sample2" priority="10" description="VSM Sample Event-2" isAlarm="true"/>
 </appext_entry>
</appext_eventdefns>
```

The file is saved with the `.xml` extension. For example: `SampleExtEventDefns.xml`.

**Create a Text File to Define the Event Names in Cisco PAM**

To define the log code names displayed in Cisco PAM, create a text file that defines a string name for each event and the event category.

In the following example, the string name for the two events and the event category are defined:

- VS_VSM_Sample1=Sample Event-1
- VS_VSM_Sample2=Sample Event-2
- Cisco_VSM=Cisco Video Surveillance Manager

The file is saved with the `.properties` extension. For example: `AppExtMessages.properties`.

---

**Recording External Events**

Chapter 2      API Functions
Import the Files into Cisco PAM

Once the XML and properties files are created, import the files into the Cisco PAM External Events module.

To do this

**Step 1** Log in to the Cisco PAM desktop application.

**Step 2** In the Events & Alarms list, choose External Events.

**Step 3** Click Import.
Recording External Events

To do this

Step 4

a. Select the XML and Properties files.
   - The XML file defines the event log codes and category for the external events.
   - The Properties file defines the text string for each code and category. The text string is the name that appears in Cisco PAM.

b. Click Next to preview the settings, or click Finish to save the changes.

For instructions to create these files, see:

- Define External Event Types Using the Event Definition Format, page 2-86
- Create a Text File to Define the Event Names in Cisco PAM, page 2-86

Step 5

(Optional) Preview the settings.

Click Finish to save the changes

Click Back to select a different file.

Step 6

The new log codes are displayed in the main External Events page.
recordExtEvent

Function name | recordExtEvent
Description | Records an external event or alarm in Cisco PAM.
Method | Post
Parameters
- secCtx—the security context object returned by authenticateUser.
- appName—the application name that calls the recordExtEvent API. This information is saved in the data field of the event.
- AcEvent—event to be recorded. The following fields are required:
  - srcDeviceName—the source device where the event originated. If the device name matches an existing entry (device of type AE.DEVICE) the event is associated with the existing device. If the device name does not exist, a new device of type AE.DEVICE with the specified name is created.
  - eventTime—the time the event occurred (in milliseconds, since epoch time).
  - type—specifies the event type for the event being recorded. This type must be imported in Cisco PAM before an event that belongs to the type can be recorded.
- srcDeviceType—the type of source device where event originated. This information is saved in the data field of the event.

Sample request

```xml
<ns2:recordExtEvent xmlns:ns2="http://cisco.com/physec/acws/"
  xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <appName>Cisco_VSM</appName>
  <event evtTime="1251242603168" type="AE.VS_VSM_Sample1">
    <srcDeviceName>FrontDomeCamera</srcDeviceName>
  </event>
</ns2:recordExtEvent>
```

Return result

Once an event is recorded, it is displayed in the Cisco PAM Event Monitoring module.

Sample response

```xml
  xmlns:ns3="http://www.w3.org/2005/08/addressing">
  <result isAnnotation="false" siteId="XFAx5kFhQ9CYGQCltUCLGg==" priority="10" evtTime="1251242603168" isAlarm="false" name="Sample Event-1" type="AE.VS_VSM_Sample1" deviceId="Uf4lTSWYSe+1RF1CRCHhCQ==" unid="159">
    <description>Sample Event-1</description>
  </result>
</ns2:recordExtEventResponse>
```

Examples: Using Java and C# Code

See Auto-Generating Java or C# Code from the WSDL File, page 1-5 for more information.

Sample Java Application Code to Record External Events

```java
// The PSIMWsService Web Service
private final static QName SERVICE = new QName("http://cisco.com/physec/acws/", "PSIMWsService");
// WSDL location URL
private final static URL WSDL_LOCATION = new URL("http://cpam-server/acws/services/psimws?wsdl");

PSIMWsService _webService = null;
```
Recording External Events

```csharp
PSIMWsPortType m_PSIMWsInterface = null;
SecurityContext _ctx = null;

try
{
    //Reference to the PSIMWsService web service
    _webService = new PSIMWsService(WSDL_LOCATION, SERVICE);

    //Get handle to PSIMWsPortType interface
    _PSIMWsInterface = _webService.getPSIMWsSoapPort();
}
catch (WebServiceException e)
{
    //Handle exception
}
catch (Exception e)
{
    //Handle exception
}

// Authenticate user
UserCredentialType uct = new UserCredentialType();
uct.setUsername("username");
uct.setPassword("password");
_ctx = _PSIMWsInterface.authenticateUser(uct);

//Call methods using interface handles
AcEvent ev = new AcEvent();
ev.setEvtTime(1234);
ev.setType("AE.APP_EVENT_1");
ev.setSrcDeviceName("deviceName");
_PSIMWsInterface.recordExtEvent(_ctx, "appName", ev, "Policy");

//In the end logout user and close the session
_PSIMWsInterface.logoutUser(_ctx);
```

Sample C# Application Code to Record External Events

```csharp
// The PSIMWsService Web Service
private String _PSIMServiceUrl = "http://cpam-server/acws/service/psimws";
private String _AccessPolicyServiceUrl = "http://cpam-server/acws/service/acpolicy";
private PSIMWsBinding _psimBinding;
private AccessPolicyBinding _apBinding;
private SecurityContext _ctx;

// PSIMWsPortType interface with SOAP/HTTP
_psimBinding = new PSIMWsBinding();
_psimBinding.Url = _PSIMServiceUrl;

// AccessPolicyPortType interface with SOAP/HTTP
```
```java
_apBinding = new AccessPolicyBinding();
_apBinding.URL = _AccessPolicyServiceUrl;

...

// Authenticate user
UserCredentialType uct = new UserCredentialType();
uct.username = "username";
uct.password = "password";
_ctx = _psimBinding.authenticateUser(uct);

...

// Call methods using interface handles
recordExtEvent ree = new recordExtEvent();
ree.secCtx = _ctx; // From authenticateUser call
ree.appName = "appName";
ree.srcDeviceType = "Policy";
AcEvent ev = new AcEvent();
ev.srcDeviceName = "deviceName";
ev.evtTime = 1234;
ev.type = "AE.APP_EVENT_1";
ree.@event = ev;

recordExtEventResponse resp = m_psimBinding.recordExtEvent(ree);

...

// In the end logout user and close the session
logoutUser lu = new logoutUser();
lu.setCtx = _ctx;
_ctx = _psimBinding.logoutUser(lu);
```
API errors return major and minor fault codes. The major fault codes are described in the following table:

<table>
<thead>
<tr>
<th>Fault Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHENTICATION_FAILED</td>
<td>User authentication failed. Invalid credentials.</td>
</tr>
<tr>
<td>COMMAND_FAILURE</td>
<td>The command could not be invoked.</td>
</tr>
<tr>
<td>CPAM_CONNECTION_FAILURE</td>
<td>The Cisco PAM connection failed.</td>
</tr>
<tr>
<td>DATA_ACCESS_EXCEPTION</td>
<td>An error occurred when accessing data in the database.</td>
</tr>
<tr>
<td>GENERIC_FAILURE</td>
<td>A generic function failure.</td>
</tr>
<tr>
<td>INCOMPATIBLE_DEV_STATE</td>
<td>The device state is incompatible for the command.</td>
</tr>
<tr>
<td>INVALID_BADGEPROPERTY</td>
<td>Invalid badge property value is specified.</td>
</tr>
<tr>
<td>INVALID_DEV_TYPE</td>
<td>Device type specified is invalid.</td>
</tr>
<tr>
<td>INVALID_METHOD_PARAM</td>
<td>An method argument contains an invalid parameter.</td>
</tr>
<tr>
<td>INVALID_PERSON_PROPERTY</td>
<td>Invalid person property value is specified.</td>
</tr>
<tr>
<td>INVALID_SECURITY_CONTEXT</td>
<td>The security context object is invalid. The object may be expired or incorrect.</td>
</tr>
<tr>
<td>METHOD_NOT_IMPLEMENTED</td>
<td>The requested method is not yet implemented.</td>
</tr>
<tr>
<td>TARGET_DEVICE_NOT_FOUND</td>
<td>The specified target device could not be found.</td>
</tr>
<tr>
<td>WSAPI_NO_LICENSE</td>
<td>The WSAPI Feature License is not found.</td>
</tr>
</tbody>
</table>
Index

recordExtEvent 189
registerAcEventCallback 136
registerAcEventXMLCallback 137
resetDoorMode 141

S
secCtx 113
security context object 113
serial number 112
setDeviceAdminState 12
setDoorMode 141
setDoorModeConfig 142
SSL 114

U
unid 115
unique ID 115
unregisterAcEventCallback 138
updateBadge 154
updatePerson 119
updateSchedule 15
updateTimeEntry 14
updateTimeEntryCollection 15
updateTimeRangeGroup 15
username and password 113

W
Web Services 18

X
X.509 certificate 114