

Pro-Watch API Service

Integration to the Pro-Watch Physical Access Control Software

HONEYWELL CONFIDENTIAL & PROPRIETARY

This work contains valuable, confidential, and proprietary information. Disclosure, use, or reproduction outside of Honeywell Inc. is prohibited except as authorized in writing. The laws protect this unpublished work. If you have received this material / document in error, please notify us immediately, and return the original material / document without making any copies.

Table of Contents

Table of Contents.....	1
Pro-Watch API Revision Log.....	2
Description	3
Visual Flow	3
Technical Notes	3
Installing the Pro-Watch API Service	5
Key settings in API configuration file	6
Enabling a Pro-Watch user	8
SOAP WSDL.....	8
Sample Program	8
Tips on troubleshooting Pro-Watch API	9
Badge Holder Access Diagram	11
Accessing the Pro-Watch hardware tree	12
REST Method Descriptions.....	13
REST Method Details.....	18
SOAP Method Descriptions.....	91
SOAP Method Return.....	95
Method Details.....	95
Subscribing to Pro-Watch Events/Alarms	135
Subscribing to Pro-Watch Data Change Events.....	140
HTTPS/SSL support in Pro-Watch DTU Service.....	143

Pro-Watch API Revision Log

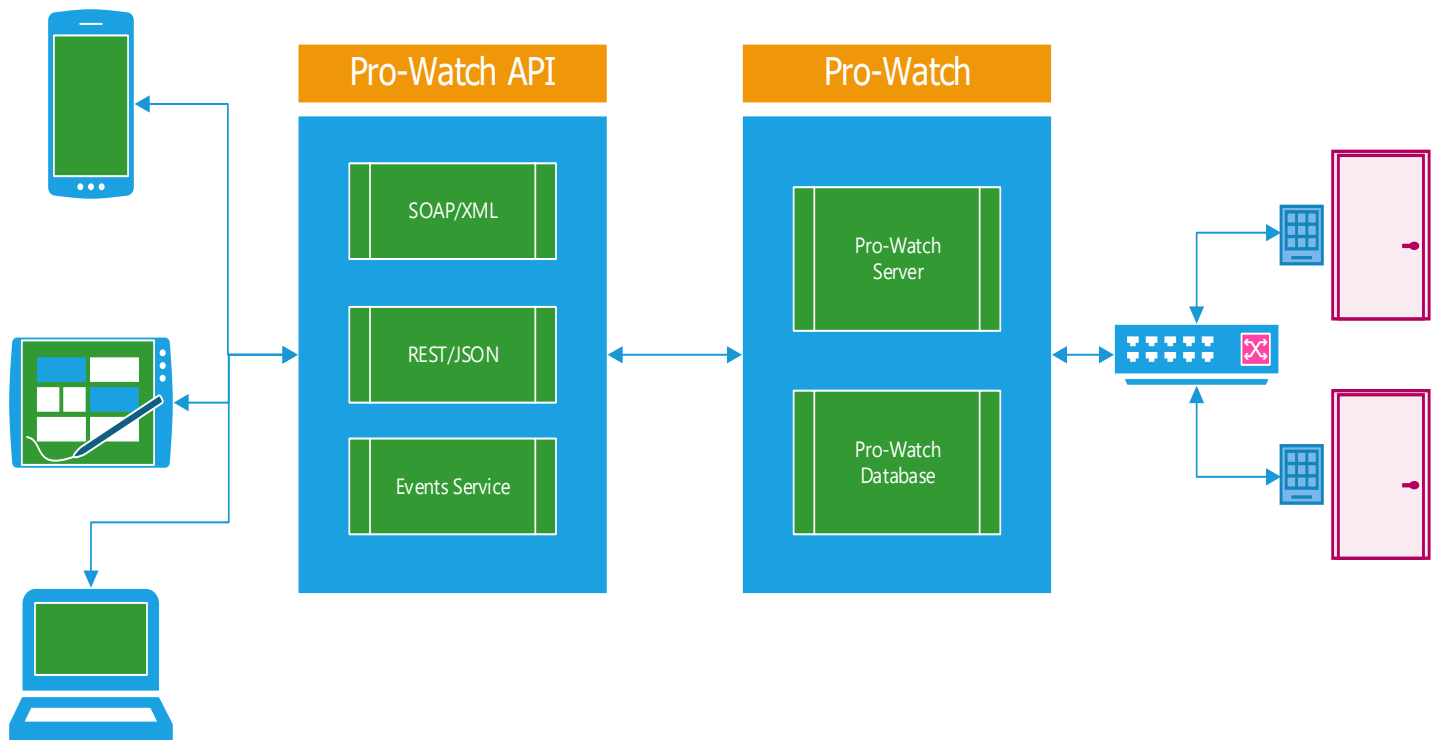
Version	Pro-Watch Version	Description
5.0.0.510	5.0 4.5.2 - 12060 4.5.1 - 11948 4.5 - 11810	<ul style="list-style-type: none"> New database change service added. This is an event based service that notifies clients when data in Pro-Watch changes Added three new Rest methods to support new database change data service and audit logging. Rest methods start with pwapi/auditlog New Postman documentation samples for all Rest calls
4.5.2.7	4.5 SP2 12060 Only	<ul style="list-style-type: none"> Added new Rest method to get all logical devices for a panel Added new Rest method to get all sub-panels for a panel Added new Rest method to get all Pro-Watch event types Modified Rest panel call to allow filtering on panel type Fixed numerous bugs when badges and card data
4.5.0.001	4.5	<ul style="list-style-type: none"> Fixed issue related to updated cards pin code, card issue date, and card expire date not automatically being downloaded to the panel. Fixed paging issue related to QueryCardsWithBadge function when only passing in badge related filters not card filters.
1.12.0.929	4.5	<ul style="list-style-type: none"> Added support for Users, and Workstations Added Sequence Number when getting Timezones Added Paging to QueryCards Fixed issue not returning notes with cards. Added QueryCardsWithBadge function that returns a list of filtered cards and all their associated badge data Added ability to perform "In" and "Not In" filtering for Strings, ID's, CardStatus's, CardTypes and BadgeTypes
1.12.0.894	4.5	Added support for Provisional Cards: <ul style="list-style-type: none"> AddProvCardByKeyField AddProvCardClearCode CopyCardToProv DeleteProvCard GetProvCards Updated QueryBadges – Added new Filter Type "In" which allows a list of values to be passed in DeleteBlob – which can be used to delete a photo or other binary object from cardholder records GetBlobTypes – returns a list of blob types Updated GetCompany and GetCompanies to include Company Address information and Clearance Code data
1.12.0.746	4.5	<ul style="list-style-type: none"> Add/update timezone Option to momentarily unlock door if card has access
1.12.0.742	4.5	<ul style="list-style-type: none"> Allowed URL's for CORS support (SignalR)

Description

The Pro-Watch API is a web service that provides a REST, hardware Event and Data change interface to the Pro-Watch access control software. The API is a Microsoft Windows service (no web server is required) that allows a client application to connect to Pro-Watch using a Pro-Watch user ID and password. Once the client application has successfully connected to the API service, all aspects of a badge can be created, modified and deleted including cards, clearance codes and door exceptions.

Connecting to the Pro-Watch API Service can be either HTTP/HTTPS.

Visual Flow



Technical Notes

The Pro-Watch API service is a self-hosted Windows service written in C# using .NET 4.8 and ASP.NET Core. In order to issue method calls, a valid Pro-Watch user and password must be passed in with the method. Actions in all method calls are controlled via the Pro-Watch class or user programs and functions. For example, to call the `PUT badges/{badgeId}` method, the user must have access in Pro-Watch to complete that action.

Installing the Pro-Watch API Service

1. Create and copy the Pro-Watch API zip file to a folder below the Pro-Watch installation directory
2. If upgrading from a previous version, run the Uninstall_PW_DTU_Service.bat file as administrator to remove the Pro-Watch DTU Service from Windows services. Delete old version.
3. In the DTU Service directory below the Pro-Watch installation directory, unzip all files to the current directory.
4. Edit the file PW-DTU-Service.exe.config file and set the endpoint and binding sections. The DTU Service is defaulted to use HTTP basic binding with no security.
5. Run the Install_PW_DTU_Service.bat file as an administrator. This will add a Windows service called Pro-Watch DTU Service.
6. In Windows Services, right click on the Pro-Watch DTU Service and select Properties.
7. Select the 'Log On' tab and add a user account to the service that has access to the Pro-Watch server and the Pro-Watch database.

Note: The DTU Service account must have access to the Pro-Watch database in SQL Server, be a valid user in Pro-Watch and have Pro-Watch access to that workstation/server. The same account that is used for Pro-Watch can and is suggested to be used for the Pro-Watch DTU service.

When using a service account please run the following command to reserve the URL (as specified in API configuration file).

Sample:

```
Netsh http add urlacl url=http://machinename:8734/pwapi user=DOMAIN\accountname
```

8. In Windows Services, start the Pro-Watch DTU Service.

Key settings in API configuration file

Pro-Watch API's configuration file - PW-DTU-Service.exe.config is in the API's installation directory. It contains flags to control features like logging, SOAP/REST/Event service interfaces, API's windows service name, using encryption etc.

Settings	Description
<pre><logFilters> <add ... enabled="true" name="Logging Enabled Filter" /> </logFilters></pre>	Enabling the log filter will help in debugging situations to capture exceptions in PWError.log file.
<pre><add key="UsePWRegistry" value="1" /> <add key="PWCommServer" value="MACHINE1" /> <add key="PWDatabaseServer" value="MACHINE1\SQL2014" /> <add key="PWDatabase" value="PWNT" /></pre>	If value is set to 1, the API will pull the Pro-Watch server, database server and database name from the registry. If set to 0 it will pick this information from settings listed.
<pre><add key="EncryptDBConnection" value="1" /> <add key="EncryptTrustServerCertificate" value="1" /></pre>	If EncryptDBConnection is set to 1, API uses encrypted connection to the database. If EncryptTrustServerCertificate is set to 0, encryption only occurs when verifiable server certificate is available. If EncryptTrustServerCertificate is set to 1, encryption always occurs but might use a self-signed certificate.
<pre><add key="ServiceDisplayName" value="Pro-Watch Web API" /> <add key="ServiceName" value="PWWebAPI" /></pre>	The following settings could be used to run multiple instances of Pro-Watch API windows service, by changing the name associated with service. This could be used on occasions where you want to run the SOAP/REST interface in its own windows service separate from the Pro-Watch Event service.
<pre><add key="StartSOAPService" value="1" /></pre>	Set StartSOAPService to 1 to turn on SOAP service. (Requires additional license)
<pre><add key="StartRESTService" value="1" /></pre>	Set StartRESTService to 1 to turn on REST service. (Requires additional license)
<pre><add key="StartEventService" value="1" /></pre>	Set StartEventService to 1 to turn on event service to subscribe to events from Pro-Watch. (Requires additional license)
<pre><add key="StartISOMService" value="0" /></pre>	Set StartISOMService to 1 to turn on ISOM based REST service Note: Currently used only by Pro-Watch Web Badging and mobile application. Requires to be run as a separate API windows service instance

<pre><system.serviceModel><services><service><host> <baseAddresses> <add baseAddress= "http://localhost:8732/ProWatch/DTUService" /> </baseAddresses> </host></service></services></pre>	<p>Represents the SOAP URL. Change localhost to machine name or IP address to allow the SOAP interfaces to be accessible on the network.</p>
<pre><add key="PWRestUrl" value="http://localhost:8734/pwapi/" /></pre>	<p>Represents the REST URL. Change localhost to machine name or IP address to allow the REST interfaces to be accessible on the network.</p>
<pre><add key="PWEventSignalRUrl" value="https://localhost:8735/" /></pre>	<p>Represents the Pro-Watch Event service URL. This SignalR based service can be used by 3rd party applications to subscribe to events from Pro-Watch over http. Change localhost to machine name or IP address to allow clients to subscribe to events over the network. Routing group rules set for the user in Pro-Watch are followed.</p>
<pre><add key="ISOMBaseURL" value="http://*:8733/" /></pre>	<p>Represents ISOM based REST service URL. Note: Currently used only by Pro-Watch Web Badging and mobile application.</p>
<pre><add key="ISOMAuthenticationScheme" value="Basic" /></pre>	<p>Please set the authentication scheme to Basic for REST/SOAP API.</p>
<pre><setting name="LogServerMessages" serializeAs="String"> <value>True</value> </setting></pre>	<p>Set the LogServerMessages to True to start anti-repudiation logs for SOAP service. The incoming and outgoing SOAP messages will be stored in Logs folder under the API's installation folder.</p>
<pre><add key="CorsOriginSettings" value="http://url1"/></pre>	<p>Set one or more comma separated URL's to be allowed as origin for CORS support.</p>

Enabling a Pro-Watch user

Requests to the Pro-Watch DTU Service must be from a valid Pro-Watch user. To enable a user to login to the Pro-Watch DTU Service:

1. In Pro-Watch, select Database Configuration
2. Select Users or Classes
3. Edit or create a user who will be connecting to the Pro-Watch DTU Service.
4. Select the Programs tab
5. Expand Database Configuration
6. Select 'User Defines'
7. Click the 'Add Function' button
8. Add 'Enable Web Password'
9. Save the User or Class record. Now the 'Web Password' for the User should be enabled.
10. Enter a 'Web Password' and save the User record.

Note: Use the Pro-Watch Username and Web Password in the authentication headers for SOAP and REST API calls.

SOAP WSDL

The Pro-Watch DTU Service WSDL file and C# proxy class is located in the DTUService directory below the Pro-Watch installation directory.

The proxy class was generated using svcutil.exe in a Visual Studio 2012 command window.

```
svcutil.exe /t:code /out:PW_DTU_Service_Proxy.cs /language:c# /config:PW_DTU_Service_App.config PW-DTU-Service.wsdl
```

Sample Program

There is a Visual Studio 2013 sample test client solution located in the DTUService directory below the Pro-Watch installation directory. The sample program connects to the Pro-Watch DTU Service, selects a badge holder, updates a badge holder and adds a new badge holder. The sample test program can also be downloaded from the FTP site.

Tips on troubleshooting Pro-Watch API

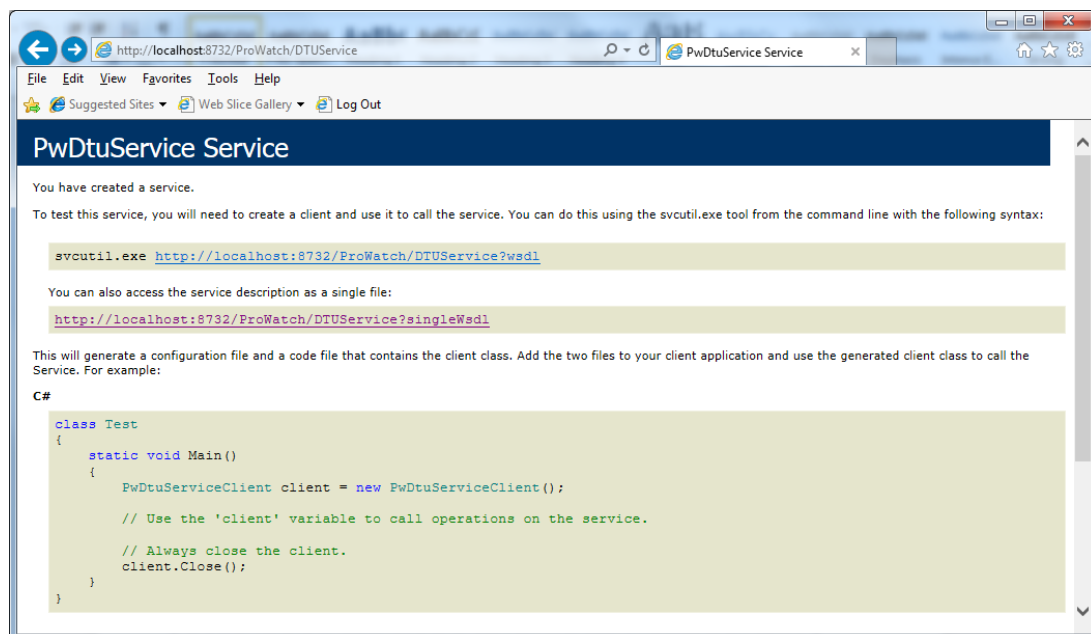
Pro-Watch API logs useful information to Windows Application logs, console and the file system. This coupled with tools like **SOAPUI** (for SOAP and REST) and **Fiddler** (REST) can be used to troubleshoot and test the API. A SOAPUI project file is included along with the API installation that provides a sample request-response template for SOAP calls and can be used to explore the API interfaces. Sample test programs can be downloaded from our FTP server.

Here are some steps to follow if your API calls are not functioning properly:

1. Make sure the required licenses have been procured and installed on the server.
 - a. **Data Transfer Utility** – Required to run SOAP/REST interface
 - b. **Data Transfer Utility API** – Required to run Event service and perform hardware actions through the API.
2. Please make sure the settings in the configuration file are correct. Make sure the required interfaces are turned on, REST/SOAP/Event service URL's are correct and localhost references have been replaced by machine name or IP address and setting for authentication and encryption scheme are used appropriately.

Note: The REST service and the ISOM service cannot run on the same instance of the Pro-Watch API windows service. Please refer to sections [Key settings in API configuration file](#) and [HTTPS/SSL support in Pro-Watch DTU Service](#) for additional information

3. When Pro-Watch API service is started, it writes to the Windows Application Event log. Here you will find if the API service could connect to Pro-Watch server and if interfaces (REST/SOAP/Event Service) were started successfully. Errors encountered are also recorded.
4. If you are testing SOAP interface, see if you can get to SOAP URL set in your API config file. Default SOAP URL is <http://localhost:8732/ProWatch/DTUService>. You should see the Pro-Watch DTU Service base address page is SOAP interface is turned on in the API config file.



5. Make sure you are using the right Pro-Watch user credentials in your SOAP and REST headers and that the user has the required Programs and Functions required to carry out the task. Please refer to section [Enabling a Pro-Watch user](#) for more information.
6. Use SOAPUI or Fiddler to test your SOAP and REST interfaces respectfully. Start with simple API's like the GetVersion or GetSites API.
7. If the SOAP API works and the REST API throws a HTTP error response, then it is possible the service account running the Pro-Watch API windows service doesn't have the required privileges to listen to the port in the REST URL set in the API config file.
 - a) Stop the API service and run the Pro-Watch API service in command line mode.
Right Click on PW_DTU_WinService.exe and Run as Administrator. If the REST API's work in this mode, close the application and proceed to step b)
 - b) Use the following command to allow the windows service account that runs the API to reserve the REST URL as follows:
Netsh http add urlacl url=http://machinename:8734/pwapi user=DOMAIN\accountname

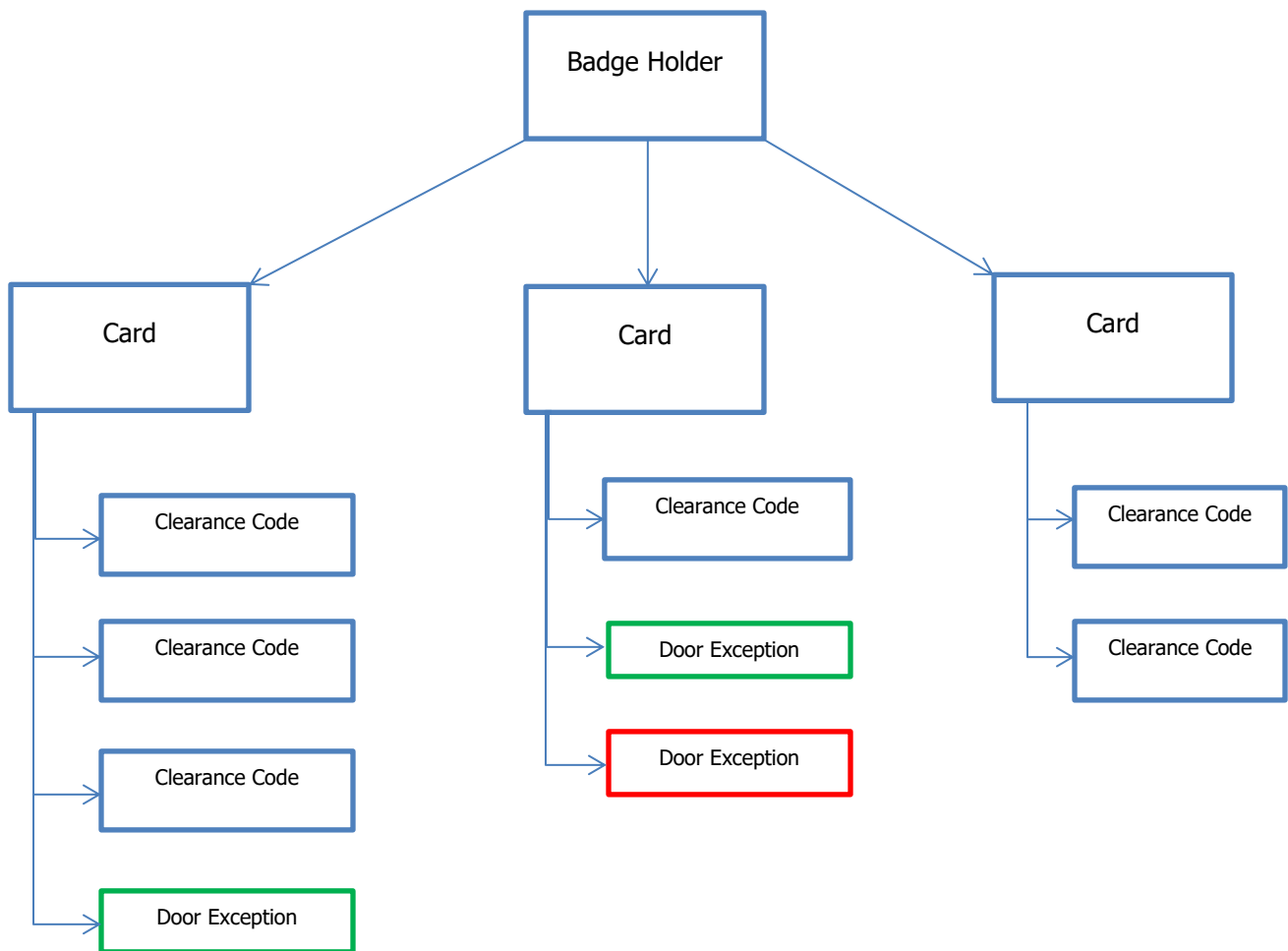
Badge Holder Access Diagram

A badge holder represents a person (Joe Blow). A card is issued to a badge holder and a badge holder can have an unlimited number of cards. Card access is made up of clearance codes (groups of doors) and door exceptions. Door exceptions can be granted (green below) and revoked (red below).

Access can be granted to a card using three methods:

1. When creating a card, a Pro-Watch company is required and that company can have multiple clearance codes assigned to it. It also can have no clearance codes assigned to it.
2. Clearance Codes (groups of doors) can be assigned to a card via company or individually.
3. Door exceptions can be assigned to a card. A door exception is a single door that is either granted or revoked to a card.

Note: To get through a door, the card must be 'Active' (all other codes are inactive), it must have an issue date greater than the current date and time, its expiration date must be greater than the current date and the card must have access to the door either through a clearance code or door exception.

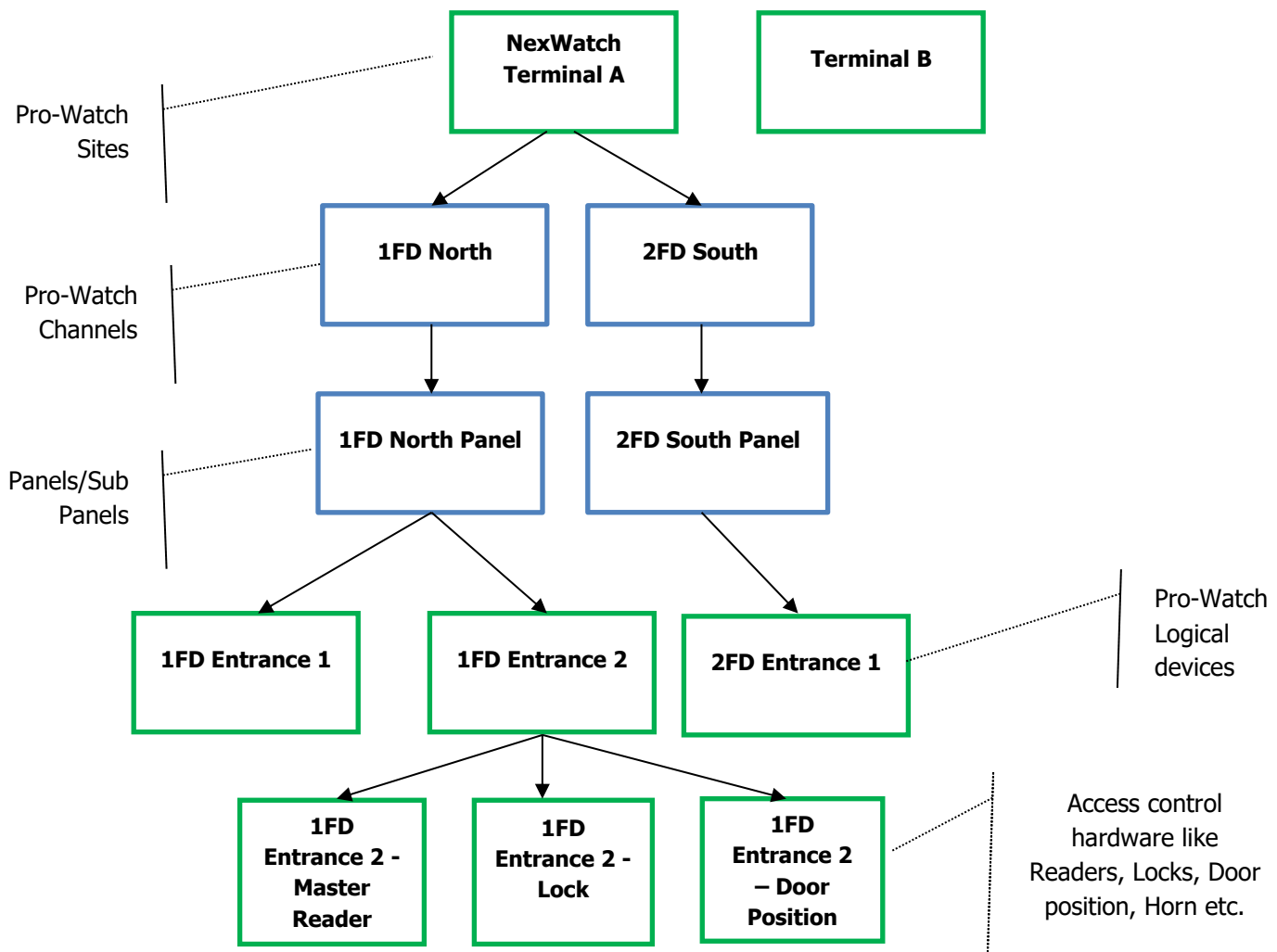


Accessing the Pro-Watch hardware tree

All hardware in Pro-Watch is organized in a hierarchical manner. Understanding this logical structure can help in locating the Hardware.

At the top of the tree is the Site. Most Pro-Watch installations name their Sites to refer to a Geographic location. Under the Site is Channel which is a logical component that holds communication parameters to communicate with Access control Hardware. Under the Channel are the Panels/Sub Panels that the Pro-Watch Server communicates with Access control hardware elements like Readers, Door Position switches, Locks, Horns etc. Since a Sub Panel could support more than one access control element there is another grouping element called Logical Device that sits between the Panel/Subpanel node and the actual Hardware. The Actual hardware sits in the leaf node of the hardware tree.

The methods available in the web service allow users to navigate the hardware tree from root node Site to the actual access control elements at the lowest level. The information obtained from these calls can be used to address the access control element. Details on Channels and Panels/Sub Panels are not available through the web service calls.



REST Method Descriptions

Action	Resource	Description
GET	/alarms	Gets all alarms in system that have not been cleared
GET	/alarms/state	Gets current dispositions for all alarms in the system
PUT	/alarms/state/change	Changes the current dispositions for alarms in the system. Workstation name needs be specified in Header (X-PW-WRKST)
PUT	/alarms/{eventId}/state/acknowledge	Acknowledge an alarm. Workstation name needs be specified in Header (X-PW-WRKST)
PUT	/alarms/{eventId}/state/clear	Clear an alarm. Workstation name needs be specified in Header (X-PW-WRKST)
PUT	/alarms/{eventId}/state/wait	Put an alarm in wait state. More details can be provided in PwWaitParam in request. Workstation name needs be specified in Header (X-PW-WRKST)
PUT	/alarms/{eventId}/state/unacknowledge	Put an alarm in unacknowledged state. Workstation name needs be specified in Header (X-PW-WRKST)
GET	/areas	Gets a list of all Areas. OData parameters are enabled
GET	/areas/{area}/occupants	Get a list of occupants in an Area. OData parameters are enabled
GET	/areas/{area}/{CardNo}/{CurrDateTime}	Get a confirmation that a card can enter the area at the specified DateTime.
GET	/assets	Get all Assets. OData parameters are enabled
POST	/assets	Add an Asset
PUT	/assets	Update an Asset
DELETE	/assets/{asset}	Delete an Asset
GET	AuditLog/Entries/{pageSize}/{page}/{startDate}/{endDate}	Gets a list of Audit Log Entries between the dates specified using paging.
GET	AuditLog/Published/{pageSize}/{page}/{startDate}/{endDate}	Gets a list of Published Audit Log Entries between the dates specified using paging
GET	AuditLog/Published/Tables	Gets a list of tables marked as being published to the data service
GET	/badgefields	Get a list of all badge fields from BADGE and BADGE_V tables. OData parameters are enabled
GET	/badgefields/{column}/dropdowns	A list of all drop down values for a drop down BADGE_V field. OData parameters are enabled

GET	/badges	Get a list of all badge holders. OData parameters are enabled
GET	/badges/{badge}	Get a Badge holder. OData parameters are enabled.
POST	/badges	Add a new badge holder
PUT	/badges	Update a badge holder
DELETE	/badges/{badge}	Delete a badge holder
GET	/badges/{badge}/cards	Get all cards for a badge holder. OData parameters are enabled
GET	/badges/{badge}/provcards	Get all provisional cards for a badge holder. OData parameters are enabled
GET	/badges/{badge}/{cards=true}	Get badge data with card data. OData parameters are enabled
GET	/badges/{badge}/photo	Get the default photo of the badge holder. Configuration file contains default photo blob ID
POST	/badges/{badge}/photo	Assign a photo (mime multipart content) to a badge holder. Configuration file contains default photo blob ID
POST	/badges/{badge}/photo/base64	Assign a photo (base64) to a badge holder. Configuration file contains default photo blob ID
GET	/badges/paging/{pagesize}/{page}	Get a page size list of all badge holders. OData parameters are enabled
POST	/badges/cards	Add a card to a badge holder
PUT	/badges/cards	Update a card on a badge holder
GET	/badges/cards/{card}	Get a card. OData parameters are enabled
DELETE	/badges/cards/{card}	Delete a Card from a badge holder
DELETE	/badges/provcards/{card}	Delete a Provisional Card from a badge holder
GET	/badges/cards/{card}/badge	Get a badge holder from a card number. OData parameters are enabled
GET	/badges/cards/{card}/badge/{card=true}	Get a badge holder with card data from a card number. OData parameters are enabled
POST	/badges/cards/copy	Create a new card for a badge holder by copying the access from another card
POST	/badges/cards/copytoprov	Create a new provional card for a badge holder by copying the access from another card
POST	/badges/cards/{card}/clearcodes	Add a list of clearance codes to add to a card
POST	/badges/provcards/{card}/clearcodes	Add a list of clearance codes to add to a provisional card
DELETE	/badges/cards/{card}/clearcodes	Delete all clearance codes from a card
DELETE	/badges/cards/{card}/clearcodes/{clearcode}	Delete a clearance code from a card

POST	/badges/cards/logdevs	Add a door/logical device exception to a card
PUT	/badges/cards/logdevs	Update a door/logical device exception to a card
DELETE	/badges/cards/{card}/logdevs/{logdev}	Delete a door/logical device from a card
GET	/badges/cards/paging/{pagesize}/{page}	Get a page size list of all badge holders, their cards, clearance codes, door exceptions and partitions. OData parameters are enabled
GET	/badges/key/{key}/{value}	Get a badge holder using a value from a BADGE_V column. OData parameters are enabled
PUT	/badges/key/{key}/{value}	Update a badge holder using a value from a BADGE_V column (Createifnotfound=false)
POST	/badges/key/{key}/{value}/cards	Add a card to a badge holder using a key field from BADGE_V table. An example would be an EMPLOYER_ID value
POST	/badges/key/{key}/{value}/provcards	Add a card to a badge holder using a key field from BADGE_V table. An example would be an EMPLOYER_ID value
GET	/badges/key/{key}/{value}/photo	Get a badge holder photo using a value from a BADGE_V column. Configuration file contains default photo blob ID.
GET	/badges/key/{key}/{value}/blobs/{type}	Get a badge holder blob (photo, signature, scanned document) using a value from a BADGE_V column
PUT	/badges/assets	Update an Asset for a badge holder
POST	/badges/assets	Add an Asset to a badge holder
GET	/badges/{badge}/assets	Get all Assets for a badge holder. OData parameters are enabled
DELETE	/badges/{badge}/assets/{asset}	Delete an Asset from a badge holder
POST	/badges/certifications/	Add a certification to a badge holder
PUT	/badges/certifications	Update a badge holder certifications
GET	/badges/{badge}/certifications	Get all certifications for a badge holder
DELETE	/badges/{badge}/certifications/{badgeID}/{rowID}	Delete a certification from a badge holder
GET	/badges/blobs/paging/{pagesize}/{page}	Get a page size list of badge holder photos. Configuration file contains default photo blob ID
GET	/badges/modified/{startDate}/{endDate}	Gets all badge holders (with card data) whose data has been modified between the start and \end dates.
POST	/badges/{badge}/partitions	Add a list of partitions to a badge holder
DELETE	/badges/{badge}/partitions/{partitionID}	Delete a partition from a badge holder

PUT	/badges/terminate/{id}	Using the unique ID and the IDColumn from the config file, this function sets BADGE.BADGE_STATUS= 'Terminated', BADGE.EXPIRE_DATE = current date, terminates all active cards and sets BADGE_C.EXPIRE_DATE = current date
GET	/badgestatuses	Get all badge statuses. OData parameters are enabled
GET	/badgetypes	Get all badge types. OData parameters are enabled
GET	/blobtypes	Get List of Blob Types
DELETE	/blobs/{badge}/{blobtype}	Delete an blob from a badge holder
GET	/certifications	Get all certifications. OData parameters are enabled
GET	/channels	Get all channels. OData parameters are enabled
GET	/clearcodes	Get all clearance codes. OData parameters are enabled
GET	/clearcodes/{clearcode}/logdevs	Get a list of logical devices in a clearance code. OData parameters are enabled
GET	/companies	Get all companies. OData parameters are enabled
POST	/events/channel	Issue a channel event
POST	/events/control	Issue a control event
POST	/events/input	Issue a input event
POST	/events/output	Issue a output event
POST	/events/panel	Issue a panel event
POST	/events/reader	Issue a reader event
POST	/events/subpanel	Issue a subpanel event
GET	/hwclass/{hwclass}/logdevs	Get all logical devices in a hardware class. OData parameters are enabled
GET	/logdevs	Get all logical devices. OData parameters are enabled
GET	/logdevs/{logdev}/hardware	Get the hardware for a logical device. OData parameters are enabled
GET	/logdevs/{logdev}/{card}/{datetime}/{unlock=false}	Gets status of card access at logical device for specified time and issue momentary unlock on door (if required).
POST	/logdevs/{logdev}/lock	Lock a logical device/door
POST	/logdevs/{logdev}/momentaryunlock	Momentary unlock a logical device/door
POST	/logdevs/{logdev}/timeoverride/{second}	Time override a door
POST	/logdevs/{logdev}/unlock	Unlock a logical device/door

POST	/logdevs/{logdev}/reenable	Reenable a logical device/door
GET	/panels	Gets all panels. OData parameters are enabled
GET	/panels/{panel}/timezones	Get all time zones for a panel. OData parameters are enabled
GET	/partitions	Get partitions. OData parameters are enabled
GET	/profiles/{badge_prof}	Get a Badge Profile. OData parameters are enabled
GET	/programs	Get program and function access in Pro-Watch
GET	/sites	Get sites. OData parameters are enabled
GET	/sites/{site}/channels	Get all channels for a site. OData parameters are enabled
GET	/sites/{site}/logdevs	Get all logical devices for a site. OData parameters are enabled
GET	/sites/{site}/panels	Get all panels for a site. OData parameters are enabled
GET	/sites/{site}/subpanels	Get all subpanels for a site. OData parameters are enabled
GET	/sites/{site}/summary	Get a site summary for counts. OData Parameters are enabled.
GET	/subpanels	Get all subpanel. OData parameters are enabled
GET	/timezones	Get timezones. OData parameters are enabled
POST	/timezones	Add a timezone and schedule
PUT	/timezones/{timezoneID}	Update an existing timezone and schedule
GET	/users	Gets all users. OData parameters are enabled
POST	/users	Adds a user using PwUser object
PUT	/users	Updates a user using PwUser object
DELETE	/users/{userID}	Deletes a user with the specified user ID
GET	/users/{userID}/partitions	Gets a list of a users partitions
POST	/users/{userID}/partition/{partitionID}	Adds a partition to a user
DELETE	/users/{userID}/partition/{partitionID}	Deletes a partition from a user
GET	/users/{userID}/workstations	Gets a list of a users workstations
POST	/users/{userID}/workstations/{workstationID}	Adds a workstation to a user
DELETE	/users/{userID}/workstations/{workstationID}	Deletes a workstation from a user
GET	/workstations	Gets a list of workstations
POST	/workstations	Adds a workstation

PUT	/workstations/{wrkstID}	Updates a workstation
DELETE	/workstations/{wrkstID}	Deletes a workstation
Get	/workstations/{wrkstID}\partitions	Gets a list of partitions for a workstation
POST	/workstations/{wrkstID}/partition/{partitionID}	Adds a partition to a workstation
DELETE	/workstations/{wrkstID}/partition/{partitionID}	Deletes a partition from a workstation
GET	/version	Get the Pro-Watch server and Pro-Watch API version

REST Method Details

Note: In the sample below, all calls were made to an http configured, basic Pro-Watch RESTful API should utilize basic authentication, issued "pre-emptively" with a valid user/pass. It works the same with https/ssl configuration.

KEY

##. METHOD /resource Text of method description in **blue**.

Parameters: Parameter definitions for required information {bracket values}, OPTIONAL values, and "Headers" in **black**.

Returns: Response from API (200 OK, 201 Created, etc.) also in **black**.

ODATA Parameter: ODATA options and examples in **brown**.
 ?\$Real Example of tested ODATA filter

//errors, bugs or non-functional items that may exist in **red** text.
 //other required element descriptions may also be in **red**.

Example Method in **green**:
 GET, POST, PUT, DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/resource |
 HEADER: "If required"
 Required Application/json media to supply or data Returns:
 (an actual example of the data structure, including brackets and quotation)

```
[
  {
    "Object": "String in quotes GUID",
    "Object Description": "String in quotes Description",
    "Complex Object1": {
      "Object1 GUID": "String in quotes GUID",
      "Object1 AdditionalItem": "Additional Item String in quotes" or option with no quotes
    },
    "DateTime": "DateTime format in quotes",
    "Bool or Num": option with no quotes in dark blue
  }
]
```

] (Notes about the data or elements in **purple**)
)

GET /alarms**Gets all alarms in system that have not been cleared**

Parameters: Header must contain valid Pro-Watch Workstation name:

"Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

//Bad Request GET http://{server}:{port}/pwapi/alarms "No Alarms Found"

//DLL Patch for 1.12.0.653 available to fix

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/state |

GET http://{server}:{port}/pwapi/alarms

X-PW-WRKST: WorkstationName

RETURNS:

```
[
  {
    "EventID": "EventIDGUID",
    "EventDate": "YYYY-MM-DDTHH:MM:SS",
    "SystemDate": "YYYY-MM-DDTHH:MM:SS",
    "EventDesc": "Input point in alarm",
    "EventTypeID": "EventTypeGUID",
    "EventTypeDesc": "Forced Door",
    "EventCode": 900,
    "Priority": 12,
    "IsAlarm": true,
    "LogicalDevice": {
      "LogDevID": "LogDevGUID",
      "Description": "LogDev Description",
      "AltDescription": "LogDev Alt Description",
      "Location": "Location Text"
    },
    "HardwareID": "SITE::HARDWARE",
    "HardwareType": 9,
    "Badge": {},
    "Card": {
      "CardNumber": "{CardNumber}",      (If Applicable)
      "Company": {}
    },
    "Message": "MessageText"              (If Applicable)
  }
]
```

(Next Alarm follows for all in Workstation Routing Group)

GET /alarms/state**Gets current dispositions for all alarms in the system****Workstation name needs be specified in Header (X-PW-WRKST)**

Parameters: Header must contain valid Pro-Watch Workstation name:

"Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/state |

GET http://{server}:{port}/pwapi/alarms/state

X-PW-WRKST: WorkstationName

```
[
  {
```

```

    "WaitIndefinitelyFlag": false,
    "WaitTime": 0,
    "EventID": "EVLOG_RID GUID",
    "EventDesc": "Input point in alarm", (Event Description)
    "EventCode": 900, (Global event code number – 900=input in alarm)
    "AlarmState": 1,
    "EventTypeID": "EventTypeID GUID",
    "EventTypeDesc": "Forced Door", (EventCode 900 on Reader = 'Forced Door')
    "HardwareID": "HardwareID Text", (e.g. "PW-5000::05010005000900")
    "HardwareType": 9,
    "DispositionDate": "YYYY-MM-DDTHH-MM-SS.sss",
    "EventProcessingFlags": 5259408,
    "User": {
      "UserID": "UserID GUID",
      "Description": "User Description",
      "EMail": null,
      "FirstName": null,
      "LastName": null,
      "UserName": null
    },
    "Workstation": {
      "WorkstationID": "WorkstationID GUID",
      "Description": "Workstation Description Text"
    }
  } (List continues for all alarms with state in Pro-Watch)
]

```

PUT /alarms/state/change **Changes the current dispositions for alarms in the system**
Workstation name needs be specified in Header (X-PW-WRKST)

Parameters: Header must contain valid Pro-Watch Workstation name:
 "Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/state/change |

PUT http://{server}:{port}/pwapi/alarms/state/change

X-PW-WRKST: WorkstationName

PUT /alarms/{eventId}/state/acknowledge **Acknowledge an alarm.**
Workstation name needs be specified in Header (X-PW-WRKST)

Parameters: {eventId} = EVLOG_RID of event to be acknowledged (starts with 0x0071)
 In addition to Authorization, Header must contain valid Pro-Watch Workstation name:
 "Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/{EVLOG_RID}/state/acknowledge |

PUT http://{server}:{port}/pwapi/alarms/0x0071.../state/acknowledge

X-PW-WRKST: WorkstationName

PUT /alarms/{eventId}/state/clear **Clear an alarm.**

Workstation name needs be specified in Header (X-PW-WRKST)

Parameters: {eventId} = EVLOG_RID of event to be cleared (starts with 0x0071)
 In addition to Authorization, Header must contain valid Pro-Watch Workstation name:
 "Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}
 (This action removes this {eventId} from the UNACK_AL table in PWNT)

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/{EVLOG_RID}/state/clear |

PUT http://{server}:{port}/pwapi/alarms/0x0071.../state/clear

X-PW-WRKST: WorkstationName

PUT /alarms/{eventId}/state/wait

Put an alarm in wait state.

More details can be provided in PwWaitParam in request.

Workstation name needs be specified in Header (X-PW-WRKST)

Parameters: {eventId} = EVLOG_RID of event to be cleared (starts with 0x0071)
 PwWaitParam must specify the duration of the wait until the event is "unacknowledged".
 In addition to Authorization, Header must contain valid Pro-Watch Workstation name:
 "Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/{EVLOG_RID}/state/wait |

PUT http://{server}:{port}/pwapi/alarms/0x0071.../state/wait

X-PW-WRKST: WorkstationName

PwWaitParam: //header parameter required for activity

```
{
  'WaitIndefinitelyFlag': false
  'WaitTime': '12' (is minutes for wait)
}
```

PUT /alarms/{eventId}/state/unacknowledge Put an alarm in unacknowledged state.

Workstation name needs be specified in Header (X-PW-WRKST)

Parameters: {eventId} = EVLOG_RID of event to be unacknowledged (starts with 0x0071)
 In addition to Authorization, Header must contain valid Pro-Watch Workstation name:
 "Header" = X-PW-WRKST | "Value" = {Machine name of Pro-Watch Workstation}

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/alarms/{EVLOG_RID}/state/unacknowledge |

PUT http://{server}:{port}/pwapi/alarms/0x0071.../state/unacknowledge

X-PW-WRKST: WorkstationName

GET /areas Gets a list of all Areas. OData parameters are enabled

Parameter: OPTIONAL areas/{logicalDevData=false} True/False option to show LogDevs.

Returns: OK - list of areas with GUID, description, and LogDev.

ODATA Parameter: Query Area Name with:
 ?\$filter=Description eq '{Area Name Text}'
 Only return data of Area Name Description with
 ?\$select=Description

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/areas
 [

```

    {
      "AreaID": "Area1 GUID",
      "Description": "Area1 Text Description",
      "LogDevs": null
    },
    {
      "AreaID": "Area2 GUID",
      "Description": "Area2 Text Description",
      "LogDevs": null
    }
  ]

```

GET /areas/{area}/occupants Get a list of occupants in an Area. OData parameters are enabled

Parameters: {area} = Area GUID

Returns: OK

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/areas/{area}/occupants
 [

```

    {
      "Area": {
        "AreaID": "Area1 GUID",
        "Description": "Area1 Text Description"
      },
      "EventTime": "Date time in YYYY-MM-DDThh:mm:ss when the occupant entered the area",
      "LogDevID": "Logical device GUID",
      "LogDevDesc": "Logical Device description",
      "BadgeID": "Badge GUID of the Badge holder in the area",
      "LastName": "Last name of the Badge holder",
      "FirstName": "First Name of the Badge holder",
      "MiddleName": "Middle Name of the Badge holder",
      "Cardnumber": "Card number of Badge holder",
      "EventID": "Event GUID that correlates to the Badge holder entering the area"
    }, (Repeats if there are more occupants)
  ]

```

GET /areas/{area}/{CardNo}/{CurrDateTime} Does a card have access to an area at the specified DateTime

Parameters: {area} = AreaID GUID
 {CardNo} = Card Number from PW Database
 {CurrDateTime} = Date & Time with format YYYY-MM-DDThh:mm:ss

Returns: OK, BAD, Unauthorized depending on card access to device

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/areas/{area}/{CardNo}/{CurrDateTime}
 Response 200 OK – if CardNo Exists
 Response 400 BAD REQUEST "Card number not found in Pro-Watch database [CardNo]" – if CardNo does not exist

Response 401 Unauthorized – if CardNo Exists but card is not allowed in area at request DateTime
 // check for - DOES HAVE ACCESS (Area) By CardNumber and Current DateTime

GET /assets **Get a list of all system Assets. OData Parameters are Enabled.**

Returns:

```
"AssetID": "Asset GUID",
"Description": "Text Description"
```

ODATA Parameter: Query Description with:
 ?\$filter= Description eq '{Description Text}'
 Only return Description data with
 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/assets |

```
[
  {
    "AssetID": "GUID",
    "Description": "Master Key"
  },
  {
    "AssetID": "GUID",
    "Description": "Encrypted Hard Drive"
  },
  {
    "AssetID": "GUID",
    "Description": "Surface Tablet"
  }
]
```

POST /assets **Add an Asset**

Parameters:

POST for assets only requires a text description of the asset.

RETURNS:

"Created" with new data:

```
"ResourceID": "New Asset GUID",
"TransStatus": 0,
"TransStatusText": null,
"TransNumber": 0
```

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/assets |

Media: ""application/json"" Data:

```
{
  "Description": "WebAPI Asset"
}
```

PUT /assets **Update an Asset**

Parameters:

PUT for assets requires the AssetID GUID and the text update

RETURNS:

OK

Example Method:

PUT |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/assets` |

Media: `"application/json"` Data:

```
{
  "AssetID": "GUID",
  "Description": "WebAPI Asset Update"
}
```

DELETE `/assets/{asset}` **Delete an Asset**

Example Method:

DELETE |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/assets/{assetID GUID}`

Returns: OK

GET `/AuditLog/Entries/{pageSize}/{page}/{startDate}/{endDate}` **Get a list of all Audit Log Entries. OData Parameters are Enabled.**

Returns:

"BatchID": "Unique ID for each BATCH of changes: Example changing multiple fields on a badge will create multiple Entries all with the same batch ID",

"TableName": "Name of the Table that had the change"

"ColumnName": "Name of the Column that had the change"

"Key1": "Primary Key of the Table that had the change"

"Key2": "additional Primary Key of the Table that had the change if multiple keys"

"Key3": "additional Primary Key of the Table that had the change if multiple keys"

"BeforeImage": "The value of the column before the change"

"AfterImage": "The value of the column after the change"

"Operation": "Add=1, Update=2, Delete=4, Report=9"

ODATA Parameter: Query Description with:
 ?`$filter=` TableName eq '{TableName Text}'
 Only return Table and Column data with
 ?`$select=` TableName, ColumnName

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/AuditLog/Entries/100/1/2020-01-14/2020-01-15` |

```
[
  {
    "BatchID": "0x000031413231414641432D434145342D3437",
    "TableName": "BADGE_V",
    "ColumnName": "BADGE_BADGENUMBER",
    "Key1": "0x002937393536354639422D303936322D3433",
    "Key2": "",
    "Key3": "",
    "BeforeImage": "",
    "AfterImage": "1234",
    "Operation": 2,
  }
]
```

```

    "DT": "2020-01-14T11:00:45.837"
  },
  {
    "BatchID": "0x000031413231414641432D434145342D3437",

    "TableName": "BADGE",
    "ColumnName": "EMAIL_PW",
    "Key1": "0x002937393536354639422D303936322D3433",
    "Key2": "",
    "Key3": "",
    "BeforeImage": "",
    "AfterImage": "john.smith@email.com",
    "Operation": 2,
    "DT": "2020-01-14T11:00:45.837"
  },
]

```

GET /AuditLog/Published/{pageSize}/{page}/{startDate}/{endDate}
all Publied Audit Log Entries. OData Parameters are Enabled.

Get a list of

Returns:

"BatchID": "Unique ID for each BATCH of changes: Example changing multiple fields on a badge will create multiple Entries all with the same batch ID",
 "TableName": "Name of the Table that had the change"
 "ColumnName": "Name of the Column that had the change"
 "Key1": "Primary Key of the Table that had the change"
 "Key2": "additional Primary Key of the Table that had the change if multiple keys"
 "Key3": "additional Primary Key of the Table that had the change if multiple keys"
 "BeforeImage": "The value of the column before the change"
 "AfterImage": "The value of the column after the change"
 "Operation": "Add=1, Update=2, Delete=4, Report=9"

ODATA Parameter: Query Description with:
 ?\$filter= TableName eq '{TableName Text}'
 Only return Table and Column data with
 ?\$select=TableName, ColumnName

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/AuditLog/Published/100/1/2020-01-14/2020-01-15 |

```

[
  {
    "BatchID": "0x000031413231414641432D434145342D3437",

    "TableName": "BADGE_V",
    "ColumnName": "BADGE_BADGENUMBER",
    "Key1": "0x002937393536354639422D303936322D3433",
    "Key2": "",
    "Key3": "",
    "BeforeImage": "",
    "AfterImage": "1234",
    "Operation": 2,
    "DT": "2020-01-14T11:00:45.837"
  },
]

```

```
{
  "BatchID": "0x000031413231414641432D434145342D3437",
  "TableName": "BADGE",
  "ColumnName": "EMAIL_PW",
  "Key1": "0x002937393536354639422D303936322D3433",
  "Key2": "",
  "Key3": "",
  "BeforeImage": "",
  "AfterImage": "john.smith@email.com",
  "Operation": 2,
  "DT": "2020-01-14T11:00:45.837"
},
]
```

GET /AuditLog/Published/Tables Get a list of all Published Audit Log Tables.

Returns: A list of tables that are marked as being published to the ProWatch Data Service. These are the tables whose changes are sent to the ProWatch Data Service as records are added / updated and deleted.

Example Method:

GET |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/AuditLog/Published/Tables> |

```
[
  "BADGE",
  "BADGE_C",
  "BADGE_CC",
  "BADGE_CERT",
  "BADGE_CL",
  "BADGE_CL",
  "BADGE_K",
  "BADGE_SCC",
  "BADGE_V",
  "BLOBS"
]
```

GET /badgefields Get a list of all badge fields from BADGE and BADGE_V tables. OData parameters are enabled

Generic list of all BADGE and BADGE_V fields in PWNT with Characteristics

ODATA Parameter: Return only Name, Length and Type results with
 ?\$select= DisplayName,FieldType,FieldLength

Return Only badgefield Types with
 ?\$expand=Asset(\$select=Description)

In order to properly filter by FieldType, use either of the following OData Methods:
 ?\$filter=(contains(cast(FieldType, 'Edm.String'),'0')) All DateTime
 ?\$filter=(contains(cast(FieldType, 'Edm.String'),'2')) All nvarchar

GET |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/badgefields> |

Returns:

```
"ColumnName": Text of Custom Column name,
"DisplayName": Descriptive name,
"FieldType": 0, (0 = DateTime SQL Type)
"ResourceType": -1,
```

"FieldLength": {number} length of SQL Field

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 1, (1 = Resource Field type, mapped to PW lookup table)

"ResourceType": 45,

"FieldLength": {number} length of SQL Field

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 2, (2 = nvarchar SQL Type)

"ResourceType": -1,

"FieldLength": {number} length of SQL Field

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 3, (3 = date SQL Field Type)

"ResourceType": -1,

"FieldLength": {number} length of SQL Field

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 4, (4 = User Defined Field Type – text values supplied)

"ResourceType": -1,

"FieldLength": {number} length of SQL Field

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 5, (5 = Blob SQL Type)

"ResourceType": -1,

"FieldLength": {number} length of SQL Field

"BlobType":

"BlobTypeID": "BlobType reference GUID",

"Description": "Description of that Blob",

"StoreFormat": 0,

"FileFormat": 0,

"BlobUser": null,

"BlobPath": null,

"BlobDate": {yyyy-mm-ddThh:mm:ss},

"EDOCType": null,

"LabelTag": null

"ColumnName": Text of Custom Column name,

"DisplayName": Descriptive name,

"FieldType": 7, (7 = Short, smallint SQL type)

"ResourceType": -1,

"FieldLength": {number} length of SQL Field

GET /badgefields/{column}/dropdowns
BADGE_V field.

A list of all drop down values for a drop down

OData parameters are enabled

Parameters: {column} = SQL Column name
(Column must be a type 4 Pro-Watch User Defined Field)

//Type 1 fields (Pro-Watch Resource Fields) do not return dropdown content.

Returns:

OK

GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badgefields/{column}/dropdowns` |

(Example from Biometric settings requirement for BADGE_V UserDefined Field for Morpho integration as described in Pro-Watch User Guide)

EXAMPLE: `GET pwapi/badgefields/MORPHO_FINGERS_1/dropdowns`

```
[
  "Left Index",
  "Left Middle",
  "Left Pinky",
  "Left Ring",
  "Left Thumb",
  "Right Index",
  "Right Middle",
  "Right Pinky",
  "Right Ring",
  "Right Thumb"
]
```

GET /badges

Get a list of all badge holders. OData parameters are enabled

//Requesting all badge holders in a Pro-Watch database can request a large amount of data returns, it is suggested that the `GET /badges/paging/{pagesize}/{page}` be used or OData \$select Filters to reduce the amount of requested data.

Parameter: {none}

Returns: OK

ODATA Parameter: Query Last Name with:
 `?$filter=LastName eq '{LNAME Text}'`
 Only return data of First Name & Last Name with
 `?$select=LastnName, FirstName`
 Return a Nested Value like BadgeStatus with
 `?$expand=BadgeStatus($select=Description)`
 Filter a Nested Value like BadgeStatus with
 `?$filter=BadgeStatus/Description eq 'ACTIVE'`

Method: GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges` |

```
[
  {
    "BadgeID": Badge GUID,
    "LastName": Text for last name,
    "FirstName": Text for first name,
    "MiddleName": Text for middle name,
    "IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
    "ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
    "BadgeType":
      "BadgeTypeID": BadgeType GUID,
      "Description": Badge Type Text

    "BadgeStatus":
      "BadgeStatusID": BadgeStatus GUID,
      "Description": "Text Description" (active/inactive are default, all others custom)
```

--Then all custom BADGE_V fields are listed, as many as exist --

"CustomBadgeFields":

"ColumnName": Text of Custom Column name,

"FieldType": 0, (0 = DateTime SQL Type)

"BooleanValue": null,

"DateValue": {yyyy-mm-ddThh:mm:ss},

"ColumnName": Text of Custom Column name,

"FieldType": 1, (1 = Resource Field type, mapped to PW lookup table)

"BooleanValue": null,

"ResourceValue": (Resources picked from a list include GUID and Description)

"ResourceID": Resource GUID,

"Description": Text Description,

"ColumnName": Text of Custom Column name,

"FieldType": 2, (2 = nvarchar SQL Type)

"TextValue": Text of the custom field,

"BooleanValue": null

"ColumnName": Text of Custom Column name,

"FieldType": 3, (3 = date SQL Field Type)

"BooleanValue": null

"ColumnName": Text of Custom Column name,

"FieldType": 4, (4 = User Defined Field Type – text values supplied)

"DropDownValue": text of selected field value, (only if selection is present)

"BooleanValue": null

"ColumnName": Text of Custom Column name,

"FieldType": 5, (5 = Blob SQL Type)

"BlobType": BlobType reference GUID,

"BooleanValue": null

"ColumnName": Text of Custom Column name,

"FieldType": 7, (7 = Short, smallint SQL type)

"ShortValue": number,

"BooleanValue": null

--END Custom BADGE_V Field Section--

"Email": Text of Email field (can be Null),

"CellPhone": Text of Cell Phone field (can be Null),

"RowVersion": Hex GUID value for row version

} (Badges continue for as many badge holders as exist in the system)

]

GET /badges/{badge} Gets all info for a badge holder. OData parameters are enabled

Parameter: {badge} = BadgeID GUID

RETURNS: OK

ODATA Parameter: Query Last Name with:

?\$filter=LastName eq '{LNAME Text}'

Only return data of First Name & Last Name with

?\$select=LastnName, FirstName

Return a Nested Value like BadgeStatus with
 ?\$expand=BadgeStatus(\$select=Description)
 Filter a Nested Value like BadgeStatus with
 ?\$filter=BadgeStatus/Description eq 'ACTIVE'

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge} |

```
[
  {
    "BadgeID": Badge GUID,
    "LastName": Text for last name,
    "FirstName": Text for first name,
    "MiddleName": Text for middle name,
    "IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
    "ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
    "BadgeType":
      "BadgeTypeID": BadgeType GUID,
      "Description": Badge Type Text

    "BadgeStatus":
      "BadgeStatusID": BadgeStatus GUID,
      "Description": "Text Description" (active/inactive are default, all others custom)
  }
]
--Then all custom BADGE_V fields are listed, as many as exist --
"CustomBadgeFields":
```

```
"ColumnName": Text of Custom Column name,
"FieldType": 0, (0 = DateTime SQL Type)
"BooleanValue": null,
"DateValue": {yyyy-mm-ddThh:mm:ss},
```

```
"ColumnName": Text of Custom Column name,
"FieldType": 1, (1 = Resource Field type, mapped to PW lookup table)
"BooleanValue": null,
"ResourceValue": (Resources picked from a list include GUID and Description)
  "ResourceID": Resource GUID,
  "Description": Text Description,
```

```
"ColumnName": Text of Custom Column name,
"FieldType": 2, (2 = nvarchar SQL Type)
"TextValue": Text of the custom field,
"BooleanValue": null
```

```
"ColumnName": Text of Custom Column name,
"FieldType": 3, (3 = date SQL Field Type)
"BooleanValue": null
```

```
"ColumnName": Text of Custom Column name,
"FieldType": 4, (4 = User Defined Field Type – text values supplied)
"DropDownValue": text of selected field value, (only if selection is present)
"BooleanValue": null
```

```
"ColumnName": Text of Custom Column name,
```



```

"FieldType": 5, (5 = Blob SQL Type)
"BlobType": BlobType reference GUID,
"BooleanValue": null

"ColumnName": Text of Custom Column name,
"FieldType": 7, (7 = Short, smallint SQL type)
"ShortValue": number,
"BooleanValue": null

```

--END Custom BADGE_V Field Section--

```

"Email": Text of Email field (can be Null),
"CellPhone": Text of Cell Phone field (can be Null),
"RowVersion": Hex GUID value for row version
}
]

```

POST /badges **Add a new badge holder**

Parameter Name = Add LastName, FirstName and any other data with format of "GET" above

Returns:

```

"BadgeID": GUID of new BadgeID created by POST command,
"AutoIncValues": null, (responds with any new Auto-Increment values)
"CardStatus": null, (if no card included – add card with POST /badges/cards)
"TransStatus": 0,
"TransStatusText": GUID of New Badge ID,
"TransNumber": 0

```

Example – Create a person with firstname, lastname, start and end date, badgetype and badge status:

Method: POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges |

Media: "application/json" Data:

```

{
  "LastName": "Webapi",
  "FirstName": "Create",
  "IssueDate": "2018-10-11T00:00:00",
  "ExpireDate": "2100-12-31T00:00:00",
  "BadgeType": {
    "Description": "STANDARD EMPLOYEE"
  },
  "BadgeStatus": {
    "Description": "ACTIVE"
  }
}

```

PUT /badges **Update an existing badge holder**

Media must include BadgeID GUID

Returns: OK

Example – Update "Create Webapi" for new FirstName, IssueDate & BadgeStatus

Method: PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges |

Media: "application/json" Data:

```

{
  "BadgeID": "{BadgeID GUID}",

```

```

"LastName": "Webapi",
"FirstName": "Created",
"IssueDate": "2018-09-11T00:00:00",
"ExpireDate": "2100-12-30T00:00:00",
"BadgeStatus": {
  "Description": "INACTIVE"
}
}

```

//Example for Custom BADGE_V Fields – Replace ColumnName “BADGE_SUPERVISOR” (et. al) text with actual Column names from BADGE_V Table.

Example – Update “Create Webapi” for new Supervisor & Supervisor Phone

Method: PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges |

Media: "application/json" Data:

```

{
  "BadgeID": "{BadgeID GUID}",
  "CustomBadgeFields": [
    {
      "ColumnName": "BADGE_SUPERVISOR",
      "TextValue": "Update Supervisor Name"
    },
    {
      "ColumnName": "BADGE_SUPERVISOR_PHONE",
      "TextValue": "502-555-0001"
    }
  ]
}

```

DELETE /badges/{badge} Delete a Badge.

Parameters: {badge} = BadgeID GUID

Returns: OK

Method: DELETE |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges{badge} |

GET /badges/{badge}/cards

Get card data from a specific badge. OData parameters are enabled

Parameters: {badge} = BadgeID GUID

Returns: OK

Card Data information from a specific badge.

ODATA Parameter: Query a specific Card with:

 ?\$filter=CardNumber eq '{CardNumber}'

Only return data of Card number, Status, Issue & Expire Dates with

 ?\$select=CardNumber,CardStatus,IssueDate,ExpireDate

Return a Nested Value like LastAccess with

 ?\$expand= LastAccess(\$select=AccessDate,LogDevDesc)

Filter a Nested Value like Cards that last accessed a "Front Door" with

 ?\$filter= LastAccess/LogDevDesc eq 'Front Door Entrance'

ODATA Only Active Cards: **In order to return only ACTIVE CARDS, Status=0, the OData filters method must be used.**

Card Status is an integer return, but can follow string filters if the field is cast as an Edm.String. Simply supplying '?\$filter=CardStatus eq '0' will not work by OData parameters. This will cause an error.

In order to properly filter by CardStatus, use either of the following OData Methods:

?\$filter=(contains(cast(CardStatus, 'Edm.String'),'0'))

?\$filter=(startswith(cast(CardStatus, 'Edm.String'),'0'))

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/cards |

```
[
  {
    "BadgeID": "0x002930123456789012345678901234567890", (Sample BadgeIDGUID)
    "CardNumber": "123456", (Sample Card Number)
    "CardStatus": 0,
    "PINCode": "",
    "IssueDate": "YYYY-MM-DDTHH:MM:SS",
    "ExpireDate": "YYYY-MM-DDTHH:MM:SS",
    "Company": {
      "ExpireMonths": 12,
      "Address1": null,
      "Address2": null,
      "City": null,
      "State": null,
      "Zip": null,
      "CompanyTypeID": null,
      "FirmExpireDate": "YYYY-MM-DDTHH:MM:SS",
      "isExpireMethodInMonths": false,
      "CompanyID": "0x004830123456789012345678901234567890",
      "Description": "Description Company",
      "ClearCodes": null,
      "Partitions": null
    },
    "CardType": {
      "BadgeTypeID": "0x002DF0123456789012345678901234567890",
      "Description": "Description CardType"
    },
    "TraceCard": false,
    "PINExempt": false,
    "GuardTour": false,
    "ADA": false,
    "VIP": false,
    "UserLevel": 0,
    "AutoDisableDays": 0,
    "UseCount": false,
    "NumberOfAttempts": 0,
    "ParadeText": null,
    "CardNotes": "",
    "ReturnDate": null,
    "CardNumberExt": "",
    "LastPrintDate": null,
    "PrintCount": 0,
    "LastChangeDate": null,
  }
]
```

```

"CustomCardStatus": null,
"LastAccess": {
  "AccessDate": "YYYY-MM-DDTHH:MM:SS",
  "LogDevID": "0x006F60123456789012345678901234567890",
  "LogDevDesc": "Description LogDev"
},
"ClearanceCodes": [ {
  "BadgeID": "0x002930123456789012345678901234567890",
  "CardNumber": "123456",
  "ClearCode": {
    "ClearCodeID": "0x004730123456789012345678901234567890",
    "Description": "Description ClearCode",
    "Description2": "",
    "Description3": "",
    "ClearCodeType": 0,
    "TimeZone": {
      "TimeZoneID": null,
      "Description": ""
    }
  },
  "LogDevs": [
    {
      "ClearCodeID": "0x004730123456789012345678901234567890",
      "LogDev": {
        "LogDevID": "0x006F10123456789012345678901234567890",
        "Description": "Description Reader"
      },
      "TimeZone": {
        "TimeZoneID": "0x000E40123456789012345678901234567890",
        "Description": "Description TimeZone"
      }
    }
  ] (Continues for all additional ClearCodes and LogDevs, including exceptions)
},
"ElevatorOutputs": [],
},
"CustomClearance": false
}],
"RowVersion": "0x0000000000042EFBE",
"ApplyCompanyClearanceCodes": null,
"CreateDate": "YYYY-MM-DDTHH:MM:SS"
}
] (And then the next Card follows)

```

GET /badges/{badge}/provcards

Get provisional card data from a specific badge. OData parameters are enabled

Parameters: {badge} = BadgeID GUID

Returns: OK

Card Data information from a specific badge.

ODATA Parameter: Query a specific Card with:

?\$filter=CardNumber eq '{CardNumber}'

Only return data of Card number, Status, Issue & Expire Dates with

?\$select=CardNumber,CardStatus,IssueDate,ExpireDate

Return a Nested Value like LastAccess with

?\$expand= LastAccess(\$select=AccessDate,LogDevDesc)

Filter a Nested Value like Cards that last accessed a "Front Door" with
 ?\$filter= LastAccess/LogDevDesc eq 'Front Door Entrance'

ODATA Only Active Cards: In order to return only ACTIVE CARDS, Status=0,
 the OData filters method must be used.
 Card Status is an integer return, but can follow string filters if the field is cast as
 an Edm.String. Simply supplying '?\$filter=CardStatus eq '0' will not work by
 OData parameters. This will cause an error.

In order to properly filter by CardStatus, use either of the following OData Methods:
 ?\$filter=(contains(cast(CardStatus, 'Edm.String'),'0'))
 ?\$filter=(startswith(cast(CardStatus, 'Edm.String'),'0'))

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/provcards |

```
[
  {
    "BadgeID": "0x002930123456789012345678901234567890", (Sample BadgeIDGUID)
    "CardNumber": "123456", (Sample Card Number)
    "CardStatus": 0,
    "PINCode": "",
    "IssueDate": "YYYY-MM-DDTHH:MM:SS",
    "ExpireDate": "YYYY-MM-DDTHH:MM:SS",
    "Company": {
      "ExpireMonths": 12,
      "Address1": null,
      "Address2": null,
      "City": null,
      "State": null,
      "Zip": null,
      "CompanyTypeID": null,
      "FirmExpireDate": "YYYY-MM-DDTHH:MM:SS",
      "isExpireMethodInMonths": false,
      "CompanyID": "0x004830123456789012345678901234567890",
      "Description": "Description Company",
      "ClearCodes": null,
      "Partitions": null
    },
    "CardType": {
      "BadgeTypeID": "0x002DF0123456789012345678901234567890",
      "Description": "Description CardType"
    },
    "TraceCard": false,
    "PINExempt": false,
    "GuardTour": false,
    "ADA": false,
    "VIP": false,
    "UserLevel": 0,
    "AutoDisableDays": 0,
    "UseCount": false,
    "NumberOfAttempts": 0,
    "ParadeText": null,
    "CardNotes": "",
    "ReturnDate": null,
  }
]
```

```

"CardNumberExt": "",
"LastPrintDate": null,
"PrintCount": 0,
"LastChangeDate": null,
"CustomCardStatus": null,
"LastAccess": {
  "AccessDate": "YYYY-MM-DDTHH:MM:SS",
  "LogDevID": "0x006F60123456789012345678901234567890",
  "LogDevDesc": "Description LogDev"
},
"ClearanceCodes": [ {
  "BadgeID": "0x002930123456789012345678901234567890",
  "CardNumber": "123456",
  "ClearCode": {
    "ClearCodeID": "0x004730123456789012345678901234567890",
    "Description": "Description ClearCode",
    "Description2": "",
    "Description3": "",
    "ClearCodeType": 0,
    "TimeZone": {
      "TimeZoneID": null,
      "Description": ""
    }
  },
  "LogDevs": [
    {
      "ClearCodeID": "0x004730123456789012345678901234567890",
      "LogDev": {
        "LogDevID": "0x006F10123456789012345678901234567890",
        "Description": "Description Reader"
      },
      "TimeZone": {
        "TimeZoneID": "0x000E40123456789012345678901234567890",
        "Description": "Description TimeZone"
      }
    }
  ] (Continues for all additional ClearCodes and LogDevs, including exceptions)
},
"ElevatorOutputs": [],
},
"CustomClearance": false
}],
"RowVersion": "0x0000000000042EFBE",
"ApplyCompanyClearanceCodes": null,
"CreateDate": "YYYY-MM-DDTHH:MM:SS",
"ProvisionDesc": "Description of Provisional Card"
}
] (And then the next Card follows)

```

GET /badges/{badge}/{cards=true}**Get badge data with card data. OData parameters are enabled**

Parameters: {badge} = BadgeID GUID
 {Cards=true} – parameter value to add card data to badge result.
 {cards=false} – also will work, but is implied by leaving parameter out of request.

Returns: OK

Badge data with Card Data added to the end.

ODATA Parameter: Query Last Name with:
 ?\$filter=LastName eq '{LNAME Text}'
 Only return data of First Name & Last Name with
 ?\$select=LastnName, FirstName
 Return a Nested Value like BadgeStatus with
 ?\$expand=BadgeStatus(\$select=Description)
 Filter a Nested Value like BadgeStatus with
 ?\$filter=BadgeStatus/Description eq 'ACTIVE'

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/{cards=true} |

```
[{
  "BadgeID": {Badge GUID},
  "LastName": "Text for last name",
  "FirstName": "Text for first name",
  "MiddleName": "Text for middle name",
  "IssueDate": "Badge Issue Date {yyyy-mm-ddThh:mm:ss}",
  "ExpireDate": "Badge Expire Date {yyyy-mm-ddThh:mm:ss}",
  "BadgeType":
    "BadgeTypeID": "{BadgeType GUID}",
    "Description": "Badge Type Text"
  "BadgeStatus":
    "BadgeStatusID": BadgeStatus GUID,
    "Description": "Text Description" (active/inactive are default, all others custom)
  --Then all custom BADGE_V fields are listed, as many as exist --
  "CustomBadgeFields":
```

```
    "ColumnName": Text of Custom Column name,
    "FieldType": 0, (0 = DateTime SQL Type)
    "BooleanValue": null,
    "DateValue": {yyyy-mm-ddThh:mm:ss}, (And additional Custom Badge Fields)
  "Email": Text of Email field (can be Null),
  "CellPhone": Text of Cell Phone field (can be Null),
  "RowVersion": Hex GUID value for row version
  "Cards":
    "BadgeID" = Badge GUID of Card Holder
    "CardNumber" = Number entered
    "CardStatus" = {0, 1, 2, 3, ..., n} for reference values of status
    "PINCode" = {PIN Number}
    "IssueDate" = Card Issue Date {yyyy-mm-ddThh:mm:ss}
    "ExpireDate" = Card Expire Date {yyyy-mm-ddThh:mm:ss}
    "Company" = (group of company data for Card Company - 13 data fields)
    "Card Type":
      "BadgeTypeID" = BadgeType GUID
      "Description" = Description Text (And additional Card Fields, see
                                     get badges/cards/{card} below)
```

GET /badges/{badge}/photo

Get the default photo of the badge holder. Configuration file contains default photo blob ID

Parameters: {badge} = BadgeID GUID of badge holder for photo request.
 PW-API Config File contains display photo default blob ID
 <!-- default blob type for photo -->

```

Returns:      <add key="PhotoBlobType" value="{BlobID GUID}" />
              OK
GET |
Endpoint: http://{Server}:{RestPort} |
Resource: /pwapi/badges/{badge}/photo |
"/{BLOB Info}/"

```

This response would require a rendering code to be used to display the image file.

POST /badges/{badge}/photo

Assign a photo (mime multipart content) to a badge holder. Configuration file contains default photo blob ID

```

Parameters:  {badge} = BadgeID GUID of badge holder for photo request.
              PW-API Config File contains display photo default blob ID
              <!-- default blob type for photo -->
              <add key="PhotoBlobType" value="{BlobID GUID}" />
Returns:      CREATED
POST |
Endpoint: http://{Server}:{RestPort} |
Resource: /pwapi/badges/{badge}/photo |
{
    "/{BLOB Info}/"
}

```

POST /badges/{badge}/photo/base64

Assign a photo (base64) to a badge holder. Configuration file contains default photo blob ID

```

Parameters:  {badge} = BadgeID GUID of badge holder for photo request.
              PW-API Config File contains display photo default blob ID
              <!-- default blob type for photo -->
              <add key="PhotoBlobType" value="{BlobID GUID}" />
Returns:      CREATED
POST |
Endpoint: http://{Server}:{RestPort} |
Resource: /pwapi/badges/{badge}/photo |
{
    "/{BLOB Info}/"
}

```

GET /badges/paging/{pagesize}/{page}

GET a list of badgeholders by pagesize and page number. OData Parameters are enabled

```

Parameters:  {pagesize} = Number of the quantity of badges per page
              {page} = Page number to request in the pages of results in quantity of {pagesize}
Returns:      OK
ODATA Parameter:  Query Last Name with:
                  ?$filter=LastName eq '{LNAME Text}'
                  Only return data of First Name & Last Name with
                  ?$select=LastnName, FirstName
Example Method:  Request '25' badges per page and send results of page '10'
GET |
Endpoint: http://{Server}:{RestPort} |
Resource: /pwapi/badges/paging/{pagesize}/{page}|
GET http://{server}:{port}/badges/paging/25/10
RETURNS:
"BadgeID": Badge GUID,

```



```

"LastName": Text for last name,
"FirstName": Text for first name,
"MiddleName": Text for middle name,
"IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
"ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
"BadgeType":
  "BadgeTypeID": BadgeType GUID,
  "Description": Badge Type Text

"BadgeStatus":
  "BadgeStatusID": BadgeStatus GUID,
  "Description": "Text Description" (active/inactive are default, all others custom)
--Then all custom BADGE_V fields are listed, as many as exist --
"CustomBadgeFields":
  "ColumnName": Text of Custom Column name,
  "FieldType": 0, (0 = DateTime SQL Type)
  "BooleanValue": null,
  "DateValue": {yyyy-mm-ddThh:mm:ss} (See GET /badge above for more examples)
--END Custom BADGE_V Field Section--
"Email": Text of Email field (can be Null),
"CellPhone": Text of Cell Phone field (can be Null),
"RowVersion": Hex GUID value for row version

```

POST /badges/cards**Add a card to an existing badge holder in the database**

Parameters: {BadgeID} GUID must be defined in the media

```

CardStatus:
  Active = 0,
  Disabled = 1,
  Lost = 2,
  Stolen = 3,
  Terminated = 4,
  Unaccounted = 5,
  Void = 6,
  Expired = 7,
  AutoDisable = 8,
  CustomCardStatus = 9

```

Example – add a new card with minimum required fields

Misc. Company Fields can be omitted as CompanyID GUID will pull data from DB.

Method: POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards |

Media: "application/json" Data:

```

{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "CardStatus": 0, (Add status options)
  "PINCode": "New PIN",
  "IssueDate": "{yyyy-mm-ddThh:mm:ss}",
  "ExpireDate": "{yyyy-mm-ddThh:mm:ss}",
  "Company": {
    "CompanyID": "CompanyID GUID",
    "Description": "Company Text Description"
  },
  "CardType": {

```

```

    "BadgeTypeID": "BadgeTypeID GUID",
    "Description": "Badge Type Description"
  }
}

```

-WARNING- Card Numbers in Pro-Watch PWNT Database are unique. An attempt to add (POST) a duplicate card number will create a violation on the API and will not add the card to the badge holder.

Returns:

Violation of PRIMARY KEY constraint 'PK_BADGE_CL'.

Cannot insert duplicate key in object 'dbo.BADGE_CL'.

The duplicate key value is ({BadgeIDGUID}, CardNumber, {LogDevIDGUID assigned}).

The statement has been terminated.

PUT /badges/cards **Update an existing card in the database**

Parameters "Card Number" = {CardNo} actual number, not GUID

Example - update a pin on a card:

Method: PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards |

Media: "application/json" Data:

```

{
  "CardNumber": "Actual Number",
  "PINCode": "New Pin"
}

```

GET /badges/cards/{card} **Get a card. OData parameters are enabled**

Parameters: {Card} = Card Number in Database (actual number, not GUID)

Returns: OK

All Card Data for CardNo if exists

ODATA Parameter: Query a specific Card with:

?\$filter=CardNumber eq '{CardNumber}'

Only return data of Card number, Status, Issue & Expire Dates with

?\$select=CardNumber,CardStatus,IssueDate,ExpireDate

Return a Nested Value like LastAccess with

?\$expand= LastAccess(\$select=AccessDate,LogDevDesc)

Filter a Nested Value like Cards that last accessed a "Front Door" with

?\$filter= LastAccess/LogDevDesc eq 'Front Door Entrance'

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards/{card} |

"BadgeID" = Badge GUID of Card Holder

"CardNumber" = Number entered

"CardStatus" = {0, 1, 2, 3, ..., n} for reference values of status

"PINCode" = {PIN Number}

"IssueDate" = Card Issue Date {yyyy-mm-ddThh:mm:ss}

"ExpireDate" = Card Expire Date {yyyy-mm-ddThh:mm:ss}

"Company" = (group of company data for Card Company - 13 data fields)

"Card Type":

"BadgeTypeID" = BadgeType GUID

"Description" = Description Text

Additional Card Details - Trace, PinExempt, GuardTour, ADA, VIP, UserLevel, AutoDisableDays, UseCount, NumberOfAttempts, ParadeText, CardNotes, ReturnDate, CardNumberExt, LastPrintDate, PrintCount, LastChangeDate, CustomCardStatus

```

"LastAccess":
    "AccessDate" = {yyyy-mm-ddThh:mm:ss}
    "LogDevID" = LogDev GUID (Last Logical Device Reader Used)
    "LogDevDesc" = LogDev Description (Last Logical Device Reader Used)
"ClearanceCodes": (repeats for as many clearance codes as are assigned)
    "BadgeID" = BadgeID GUID
    "CardNumber" = CardNumber, not GUID
    "ClearCode":
        "ClearCodeID" = ClearCode GUID
        "Description" = Description ClearCode
        "Description2" = Detailed Description text
        "Description3" = Alternate Description text
        "ClearCodeType" = type (number for Clearance Code)
        "TimeZone" = Default Time Zone for Clearance Code
Logical Devices within this Clearance Code Follow (all - may be long list)
"LogDevs":
    "ClearCodeID" = ClearCode GUID
    "LogDev":
        "LogDevID" = LogDevGUID
        "Description" = LogDev Description
    "TimeZone":
        "TimeZoneID" = TimeZone GUID (Time Zone granted for access to this Logical Device Reader)
        "Description" = Description of Time Zone granted
LogDevs repeats for next reader(s) with ClearCodeID and LogDev/TimeZone
"ElevatorOutputs" = any Elevator output groups granted by clearance code
"CustomClearance" = true/false for custom clearance edits
(Clearance Codes repeat for next and remaining clearance codes on same card [starting with BadgeID,
CardNumber, etc.])
"LogDevExceptions" = Any logical device exceptions (GRANT/REVOKE per single door) added to this
CardNumber
    "BadgeID" = BadgeID GUID
    "CardNumber" = CardNumber, not GUID
    "GrantedRevoked" = Binary for either option {0=granted, 1=revoked}
    "LogDev":
        "LogDevID" = LogDevGUID
        "Description" = LogDev Description
    "TimeZone": = TimeZone of granted access to logical device {null if exception=REVOKED}
        "TimeZoneID" = TimeZone GUID
        "Description" = Description of Time Zone granted
(if temporary granted or revoked access is added, additional returns follow)
    "TempAccess" = true (only appears if true)
    "TempStartDate" = {yyyy-mm-ddThh:mm:ss}
    "TempEndDate" = {yyyy-mm-ddThh:mm:ss}

```

RESULTS EXAMPLE:

```

{
  "BadgeID": "{Badge GUID}",
  "CardNumber": "{Card Number}",
  "CardStatus": 0,
  "IssueDate": "Card Issue Date {yyyy-mm-ddThh:mm:ss}",
  "ExpireDate": "Card Expire Date {yyyy-mm-ddThh:mm:ss}",
  "Company": {
    "CompanyID": "{CompanyGUID}",
    "Description": "{CompanyTextDescription}"
  }
}

```

```

},
"CardType": {
  "BadgeTypeID": "{BadgeTypeGUID}",
  "Description": "Card Type Description"
},
"LastChangeDate": "{yyyy-mm-ddThh:mm:ss}",
"GuardTour": false,
"ADA": false,
"VIP": false,
"UserLevel": 0,
"AutoDisableDays": 0,
"UseCount": false,
"NumberOfAttempts": 0,
"ParadeText": "",
"CardNotes": "",
"TraceCard": false,
"PINExempt": false,
"LastAccess": {
  "AccessDate": "{yyyy-mm-ddThh:mm:ss}",
  "LogDevID": "{LogDevGUID}",
  "LogDevDesc": "LogDev Description"
},
"ClearanceCodes": [
  {
    "BadgeID": "{BadgeGUID}",
    "CardNumber": "{CardNumber}",
    "ClearCode": {
      "ClearCodeID": "{ClearanceCode GUID}",
      "Description": "Clearance Code Description",
      "Description2": "Clearance Code Description2",
      "Description3": "Clearance Code Description3",
      "TimeZone": {
        "Description": "Default TimeZone Description"
      },
    },
    "LogDevs": [
      {
        "ClearCodeID": "{ClearanceCode GUID}",
        "LogDev": {
          "LogDevID": "{LogDev GUID}",
          "Description": "LogDev Description"
        },
        "TimeZone": {
          "TimeZoneID": "{TimeZone GUID}",
          "Description": "LogDev Timezone Description"
        }
      }, (Continues for all additional Clearance Codes and LogDevs)
    ],
    "ElevatorOutputs": []
  }
],
"LogDevExceptions": [
  {

```

```

    "BadgeID": "{BadgeID GUID}",
    "CardNumber": "{CardNumber}",
    "LogDev": {
      "LogDevID": "{LogDev GUID}",
      "Description": "LogDev Description"
    },
    "TimeZone": {
      "TimeZoneID": "{TimeZone GUID}",
      "Description": "LogDev Timezone Description"
    }
  },
  {
    "BadgeID": "{BadgeID GUID}",
    "CardNumber": "{CardNumber}",
    "GrantedRevoked": 1,
    "LogDev": {
      "LogDevID": "{LogDev GUID}",
      "Description": "LogDev Description"
    }
  }
],
"RowVersion": "0x00000000000019CE39",
"CreateDate": "{yyyy-mm-ddThh:mm:ss}",
}

```

DELETE **/badges/cards/{card}** **Deletes a card from a badge holder by card number.**

Parameters "Card" = {CardNo} actual number, not GUID"
Returns: OK
 (Deletions are logged in the PW Audit Log)

DELETE **/badges/provcards/{card}** **Deletes a provional card from a badge holder by provisional card number.**

Parameters "Card" = {ProvCardNo} actual number, not GUID"
Returns: OK
 (Deletions are logged in the PW Audit Log)

GET **/badges/cards/{card}/badge** **Get badge data from a card number. OData parameters are enabled**

Parameters: {Card} = Card Number in Database (actual number, not GUID)
Returns: OK
 All Badge Data for badge holder of CardNo if exists

ODATA Parameter: Query

Last Name with:

 ?\$filter=LastName eq '{LNAME Text}'
 Only return data of First Name & Last Name with
 ?\$select=LastnName, FirstName

Example Method:

GET |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/badges/cards/{card}/badge> |

(See GET /badges/{badge} above for data structure)

GET /badges/cards/{card}/badge/{card=true}**Get badge data from a card number. OData parameters are enabled**

Parameters: {Card} = Card Number in Database (actual number, not GUID)
 {Cards=true} – parameter value to add card data to badge result.
 {cards=false} – also will work, but is implied by leaving parameter out of request.

Returns: OK
 All Badge and Card Data for badge holder of CardNo if exists

ODATA Parameter: Query a specific Card with:
 ?\$filter=CardNumber eq '{CardNumber}'
 Only return data of Card number, Status, Issue & Expire Dates with
 ?\$select=CardNumber,CardStatus,IssueDate,ExpireDate
 Return a Nested Value like LastAccess with
 ?\$expand= LastAccess(\$select=AccessDate,LogDevDesc)
 Filter a Nested Value like Cards that last accessed a "Front Door" with
 ?\$filter= LastAccess/LogDevDesc eq 'Front Door Entrance'

(See GET /badges/{badge}/{card=true} above for data structure)

POST /badges/cards/copy**Create a new card for a badge holder by copying the access from another card**

Parameters: Card Number must be defined in the media

Returns: "Created"

Copying a card in this method creates a new card on the Badge Holder who has ownership of the original card. The "CopyCardNumber" defines this badge holder. All attributes are copied, including PINCode, ClearanceCodes, LogDev Exceptions, Company and Card Type. The new card is automatically "Active" and ready for use. A Boolean value can leave the original card active or disable the original card.

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards/copy |

Media: ""application/json"" Data:

```
{
  "CopyCardNumber": "{OriginalCardNumber}",
  "NewCardNumber": "{NewCardNumber}",
  "IssueDate": "YYYY-MM-DDTHH:MM:SS",
  "ExpireDate": "YYYY-MM-DDTHH:MM:SS",
  "DisableCopyCard": bool      (0=leave original card active, 1=disable original card)
}
```

Real example for copying card 66778 to a new 77998 card with new dates, and disabling the original card.

POST http://{server}:{restport}/pwapi/badges/cards/copy

```
{
  "CopyCardNumber": "66778",
  "NewCardNumber": "77998",
  "IssueDate": "2018-11-11T12:45:00",
  "ExpireDate": "2019-11-10T12:45:00",
  "DisableCopyCard": 1
}
```

Card Number	PIN	Status	Company	Card Type	Issue Date	Expire Date	Last Change
77998	1234	Active	HONEYWELL IN	STANDARD EMP	11/26/2018	11/25/2019	11/26/2018
66778	1234	Disabled	HONEYWELL IN	STANDARD EMP	05/19/2010	12/13/2022	11/26/2018

POST /badges/cards/copytoprov**Create a new provisional card for a badge holder by copying the access from another card**

Parameters: Card Number must be defined in the media

Returns: "Created"

Copying a card in this method creates a new provisional card on the Badge Holder who has ownership of the original card. The "CopyCardNumber" defines this badge holder. All attributes are copied, including PINCode, ClearanceCodes, LogDev Exceptions, Company and Card Type. The new card is automatically "Active" and ready for use. A Boolean value can leave the original card active or disable the original card.

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/cards/copytoprov` |

Media: `"application/json"` Data:

```
{
  "CopyCardNumber": "{OriginalCardNumber}",
  "IssueDate": "YYYY-MM-DDTHH:MM:SS",
  "ExpireDate": "YYYY-MM-DDTHH:MM:SS",
  "DisableCopyCard": bool,          (0=leave original card active, 1=disable original card)
  "ProvisionDescription": "Any Discription to describe what this card is"
}
```

Real example for copying card 66778 with new dates, and disabling the original card.

POST `http://{server}:{restport}/pwapi/badges/cards/copytoprov`

```
{
  "CopyCardNumber": "66778",
  "IssueDate": "2018-11-11T12:45:00",
  "ExpireDate": "2019-11-10T12:45:00",
  "DisableCopyCard": 1,
  "ProvisionDescription": "BatchID – 123456789"
}
```

POST /badges/cards/{card}/clearcodes Add a list of clearance codes to add to a card

Parameters: "Card" = {CardNo} actual number, not GUID"

"ClearCodes" = List of Clearance Codes to add to the card.

This parameter can run with ClearCodeID GUID or ClearCodeDescription Text.

Returns: "Created"

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/cards/{card}/clearcodes` |

Media: `"application/json"` Data:

```
[
  {
    "ClearCodeID": "{GUID}"
  }
]
```

POST /badges/provcards/{provional card}/clearcodes Add a list of clearance codes to add to a provisional card

Parameters: "Provisional Card" = {ProvCardNo} actual number, not GUID"

"ClearCodes" = List of Clearance Codes to add to the card.

This parameter can run with ClearCodeID GUID or ClearCodeDescription Text.

Returns: "Created"

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: /pwapi/badges/provcards/{provisional card}/clearcodes |

Media: ""application/json"" Data:

```
[
  {
    "ClearCodeID": "{GUID}"
  }
]
```

DELETE /badges/cards/{card}/clearcodes **Deletes all clearance codes from a card**

Parameters: "Card" = {CardNo} actual number, not GUID"

Returns: OK

(This method does NOT delete any Logical Device Exceptions on the card.)

DELETE /badges/cards/{card}/clearcodes/{clearcode} **Delete a clearance code from a card**

Parameters: "Card" = {CardNo} actual number, not GUID"

"clearcode" = {clearcodeID} GUID of Clearance Code ID

Returns: OK

POST /badges/cards/logdevs **Add a Logical Device to a Card**

Parameters: Card Number must be defined in the media

Only one Logical Device Exception may be added at a time.

Example – add a LogDev with minimum required fields

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards/logdevs |

Media: ""application/json"" Data:

Example for Granted LogDev:

```
{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "GrantedRevoked": 0, (0 for Grant, 1 for Revoked)
  "LogDev": {
    "LogDevID": "LogDevID GUID",
    "Description": "Description Text"
  },
  "TimeZone": {
    "TimeZoneID": "TimeZoneID GUID",
    "Description": "System All Times"
  }
}
```

Example for Revoked LogDev:

```
{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "GrantedRevoked": 1, (0 for Grant, 1 for Revoked)
  "LogDev": {
    "LogDevID": "LogDevID GUID",
    "Description": "Description Text"
  },
  "TimeZone": null (LogDev Revoke has no time-zone)
}
```


PUT /badges/cards/logdevs**Update a Logical Device on a Card**

Parameters – Card Number must be defined in the media

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/cards/logdevs |

Media: "application/json" Data:

Example for Granted LogDev:

```
{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "GrantedRevoked": 0, (0 for Grant, 1 for Revoked)
  "LogDev": {
    "LogDevID": "LogDevID GUID",
    "Description": "Description Text"
  },
  "TimeZone": {
    "TimeZoneID": "TimeZoneID GUID",
    "Description": "System All Times"
  }
}
```

Example for Revoked LogDev:

```
{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "GrantedRevoked": 1, (0 for Grant, 1 for Revoked)
  "LogDev": {
    "LogDevID": "LogDevID GUID",
    "Description": "Description Text"
  },
  "TimeZone": null (LogDev Revoke has no time-zone)
}
```

DELETE /badges/cards/{card}/logdevs/{logdev} Delete a door/logical device from a card

Parameters: {card} = represents card number
 {logdev} = represents a logical device GUID

Returns: OK

GET /badges/cards/paging/{pagesize}/{page}

GET a list of badges and cards by pagesize and page number. Odata parameters are enabled.

Parameters: {pagesize} = Number of the quantity of badges per page
 {page} = Page number to request in the pages of results in quantity of {pagesize}

ODATA Parameter: Query Last Name with:
 ?\$filter=LastName eq '{LNAME Text}'
 Only return data of First Name & Last Name with
 ?\$select=LastnName, FirstName

Returns: OK

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/paging/{pagesize}/{page}|

"BadgeID": Badge GUID,

"LastName": Text for last name,

```

"FirstName": Text for first name,
"MiddleName": Text for middle name,
"IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
"ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
"BadgeType":
  "BadgeTypeID": BadgeType GUID,
  "Description": Badge Type Text

"BadgeStatus":
  "BadgeStatusID": BadgeStatus GUID,
  "Description": "Text Description" (active/inactive are default, all others custom)
--Then all custom BADGE_V fields are listed, as many as exist --
"CustomBadgeFields":

  "ColumnName": Text of Custom Column name,
  "FieldType": 0, (0 = DateTime SQL Type)
  "BooleanValue": null,
  "DateValue": {yyyy-mm-ddThh:mm:ss}, (And additional Custom Badge Fields)
--END Custom BADGE_V Field Section--
"Email": Text of Email field (can be Null),
"CellPhone": Text of Cell Phone field (can be Null),
"RowVersion": Hex GUID value for row version
"Cards":
  "BadgeID" = Badge GUID of Card Holder
  "CardNumber" = Number entered
  "CardStatus" = 0 (0, 1, 2, 3, ..., n) for reference values of status
  "PINCode" = {PIN Number}
  "IssueDate" = Card Issue Date {yyyy-mm-ddThh:mm:ss}
  "ExpireDate" = Card Expire Date {yyyy-mm-ddThh:mm:ss}
  "Company" = (group of company data for Card Company - 13 data fields)
  "Card Type":
    "BadgeTypeID" = BadgeType GUID
    "Description" = Description Text (And additional Card Fields, see
                                     GET badges/cards/{card})

```

GET /badges/key/{key}/{value}

Get a badge holder using a value from a BADGE_V column as a filter. OData Parameters are enabled.

Parameters: OPTIONAL /badges/key/{key}/{value}/{cards=true} to add card data to results
 {key} = the name of the BADGE_V Column
 {value} = the value to look for in BADGE_V

ODATA Parameter: Only return data of First Name & Last Name with
 ?\$select=LastnName, FirstName

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/key/{key} |

Media: ""application/json"" Data:

Example URL for finding a badge holder with "Employee ID" of 66778, key column name is "EMPLOYEE_ID":

"GET http://localhost:port/pwapi/badges/key/EMPLOYEE_ID/66778"

RETURNS:

```

"BadgeID": Badge GUID,
"LastName": Text for last name,

```

```

"FirstName": Text for first name,
"MiddleName": Text for middle name,
"IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
"ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
"BadgeType":
  "BadgeTypeID": BadgeType GUID,
  "Description": Badge Type Text

"BadgeStatus":
  "BadgeStatusID": BadgeStatus GUID,
  "Description": "Text Description" (active/inactive are default, all others custom)
--Then all custom BADGE_V fields are listed, as many as exist --
"CustomBadgeFields":
  "ColumnName": Text of Custom Column name,
  "FieldType": 0, (0 = DateTime SQL Type)
  "BooleanValue": null,
  "DateValue": {yyyy-mm-ddThh:mm:ss} (See GET badge above for more examples)
--END Custom BADGE_V Field Section--
"Email": Text of Email field (can be Null),
"CellPhone": Text of Cell Phone field (can be Null),
"RowVersion": Hex GUID value for row version

```

PUT /badges/key/{key}/{value}

Get a badge holder using a value from a BADGE_V column as a filter. OData Parameters are enabled.

Parameters: {key} = the name of the BADGE_V Column
 {value} = the value to look for in BADGE_V
 OPTIONAL: /badges/key/{key}/{value}/{createifnotfound=false} True/False to create a new badge holder if the key value does not exist in the database.

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/key/{key} |

Example URL for finding a badge holder with "Employee ID" of 66778, key column name is "EMPLOYEE_ID":

"PUT http://localhost:port/pwapi/badges/key/EMPLOYEE_ID/66778"

Media: ""application/json"" Data:

```

{
  "CustomBadgeFields": [
    {
      "ColumnName": "BADGE_SUPERVISOR",
      "TextValue": "Update Supervisor Name"
    },
    {
      "ColumnName": "BADGE_SUPERVISOR_PHONE",
      "TextValue": "502-555-0001"
    }
  ]
}

```

POST /badges/key/{key}/{value}/cards

Add a card to an existing badge holder in the database based on a key value (i.e. Employee ID number)

Parameters: {key} = the name of the BADGE_V Column

{value} = the value to look for in BADGE_V
Returns: OK

Example – add a new card with minimum required fields

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/key/{key}/{value}/cards |

Media: "application/json" Data:

Misc. Company Fields can be omitted as CompanyID GUID will pull data from DB.

```
{
  "BadgeID": BadgeID GUID,
  "CardNumber": "Actual Number",
  "CardStatus": 0, (Add other status options)
  "PINCode": "New PIN",
  "IssueDate": "{yyyy-mm-ddThh:mm:ss}",
  "ExpireDate": "{yyyy-mm-ddThh:mm:ss}",
  "Company": {
    "CompanyID": "CompanyID GUID",
    "Description": "Company Text Description"
  },
  "CardType": {
    "BadgeTypeID": "BadgeTypeID GUID",
    "Description": "Badge Type Description"
  }
}
```

Real Example URL for adding card "77889" to employee with "Employee ID" of 66778, key column name is "EMPLOYEE_ID":

"POST http://localhost:8744/pwapi/badges/key/EMPLOYEE_ID/66778/cards"

```
{
  "CardNumber": "77889",
  "CardStatus": 0,
  "PINCode": "4321",
  "IssueDate": "2019-01-24T15:22:45",
  "ExpireDate": "2020-10-24T15:22:45",
  "Company": {
    "CompanyID": "{Company GUID}",
    "Description": "HONEYWELL INC."
  },
  "CardType": {
    "BadgeTypeID": "{BadgeTypeID GUID}",
    "Description": "STANDARD EMPLOYEE"
  }
}
```

POST /badges/key/{key}/{value}/provcards

Add a provisional card to an existing badge holder in the database based on a key value (i.e. Employee ID number)

Parameters: {key} = the name of the BADGE_V Column
 {value} = the value to look for in BADGE_V

Returns: OK

Example – add a new card with minimum required fields

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/key/{key}/{value}/provcards` |

Media: `"application/json"` Data:

Misc. Company Fields can be omitted as CompanyID GUID will pull data from DB.

```
{
  "BadgeID": "BadgeID GUID",
  "CardNumber": "",
  "CardStatus": 0, (Add other status options)
  "PINCode": "New PIN",
  "IssueDate": "{yyyy-mm-ddThh:mm:ss}",
  "ExpireDate": "{yyyy-mm-ddThh:mm:ss}",
  "Company": {
    "CompanyID": "CompanyID GUID",
    "Description": "Company Text Description"
  },
  "CardType": {
    "BadgeTypeID": "BadgeTypeID GUID",
    "Description": "Badge Type Description"
  },
  "ProvisionDesc": "Description of the Provisional Card"
}
```

Real Example URL for adding card "77889" to employee with "Employee ID" of 66778, key column name is "EMPLOYEE_ID":

"POST `http://localhost:8744/pwapi/badges/key/EMPLOYEE_ID/66778/cards`"

```
{
  "CardNumber": "",
  "CardStatus": 0,
  "PINCode": "4321",
  "IssueDate": "2019-01-24T15:22:45",
  "ExpireDate": "2020-10-24T15:22:45",
  "Company": {
    "CompanyID": "{Company GUID}",
    "Description": "HONEYWELL INC."
  },
  "CardType": {
    "BadgeTypeID": "{BadgeTypeID GUID}",
    "Description": "STANDARD EMPLOYEE"
  },
  "ProvisionDesc": "BatchID - 123456789"
}
```

GET `/badges/key/{key}/{value}/photo`

Get the default photo of the badge holder. Configuration file contains default photo blob ID

Parameters: `{key}` = the name of the BADGE_V Column
 `{value}` = the value to look for in BADGE_V
 PW-API Config File contains display photo default blob ID
 <!-- default blob type for photo -->
 <add key="PhotoBlobType" value="{BlobID GUID}" />

Returns: OK

GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/key/{key}/{value}/photo` |

"/{BLOB Info}/"

This response would require a rendering code to be used to display the image file.

GET /badges/key/{key}/{value}/blobs/{type}

Get a badge holder blob (photo, signature, scanned document) using a value from a BADGE_V column

Parameters:

{key} = the name of the BADGE_V Column

{value} = the value to look for in BADGE_V

{type} = the GUID of the blob type ID

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/key/{key}/{value}/blobs/{BlobTypeID} |

Media: ""application/json"" Data:

Example URL for finding a badge holder with "Employee ID" of 66778, key column name is "EMPLOYEE_ID":

"GET http://localhost:8744/pwapi/badges/key/EMPLOYEE_ID/66778/blobs/{SignatureBlobID_GUID}"

RETURNS: OK

"/{BLOB Info}/"

This response would require a rendering code to be used to display the image file.

PUT /badges/assets

Update an Asset for a badge holder

Parameters:

{BadgeID} = Badge GUID to select

{AssetID} = Asset GUID of existing Asset to be added

{DateIssued} = {yyyy-mm-ddThh:mm:ss}

{DateDue} = {yyyy-mm-ddThh:mm:ss}

{DateReturned} = {yyyy-mm-ddThh:mm:ss} (can be null)

{AssetStatus} = {Assigned|Invoiced|Lost|Returned|Stolen|Unaccounted}

{AssetNote} = {Note field for text}

Returns OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/assets |

Media: ""application/json"" Data:

```
[
  {
    "BadgeID": "{BadgeIDGUID}",
    "Asset": {
      "AssetID": "{AssetIDGUID}",
    },
    "DateIssued": "{yyyy-mm-ddThh:mm:ss}",
    "DateDue": "{yyyy-mm-ddThh:mm:ss}",
    "DateReturned": null,
    "AssetStatus": "Assigned"
  }
]
```

POST /badges/assets

Add an Asset to a badge holder

Parameters:

{BadgeID} = Badge GUID to select

{AssetID} = Asset GUID of existing Asset to be added

{DateIssued} = {yyyy-mm-ddThh:mm:ss}

```

{DateDue} = {yyyy-mm-ddThh:mm:ss}
{DateReturned} = {yyyy-mm-ddThh:mm:ss} (can be null)
{AssetStatus} = {Assigned|Invoiced|Lost|Returned|Stolen|Unaccounted}
{AssetNote} = {Note field for text}

```

Returns: "Created"

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/assets` |

Media: `"application/json"` Data:

```

[
  {
    "BadgeID": "{BadgeIDGUID}",
    "Asset":
    {
      "AssetID": "{AssetIDGUID}",
    },
    "DateIssued": "{yyyy-mm-ddThh:mm:ss}",
    "DateDue": "{yyyy-mm-ddThh:mm:ss}",
    "DateReturned": null,
    "AssetStatus": "Assigned",
    "AssetNote": "Text to insert"
  }
]

```

GET /badges/{badge}/assets Get all Assets for a badge holder. OData parameters are enabled

Parameter: {badge} = Pro-Watch BadgeID GUID

ODATA Parameter: Return only Issued, Due and Status results with
 ?\$select=DateIssued,DateDue,AssetStatus
 Return a Nested Value like Asset Description with
 ?\$expand=Asset(\$select=Description)

Returns: OK

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/badges/{badge}/assets` |

"BadgeID": BadgeID GUID

"Asset":

 "AssetID": {GUID}

 "Description": {description text}

"AssetNo":

"DateIssued": {yyyy-mm-ddThh:mm:ss}

"DateDue": {yyyy-mm-ddThh:mm:ss}

"DateReturned": {yyyy-mm-ddThh:mm:ss} (can be null)

"AssetStatus": {Assigned|Invoiced|Lost|Returned|Stolen|Unaccounted}

"AssetNote": Text of note entered

--Additional Assets follow for complete list.

DELETE /badges/{badge}/assets/{asset} Delete an Asset from a badge holder

Parameter: {badge} = BadgeID GUID 0x0029... |

 {asset} = AssetID GUID 0x002E... |

 OPTIONAL = {number = ' '} which denotes the number of the asset key.

Example Method:

DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/badges/{badge}/assets/{asset}/{number = "} |

POST /badges/certifications **Add a certification to an existing badge holder**

Parameters: {BadgeID} GUID must be defined in the media
 {Certification} GUID must be defined in the media
 Returns: "Created"

Example Method:

POST |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/badges/certifications |
 Media: ""application/json"" Data:

//Minimum required data to post a cert:

```
{
  "BadgeID": "{BadgeID GUID}",
  "Certification": {
    "CertificationID": "{CertID GUID}",
  },
  "CertDate": "{yyyy-mm-ddThh:mm:ss}"
}
```

//Full available data POST options:

```
POST |
Endpoint: http://{Server}:{RestPort} |
Resource: /pwapi/badges/certifications |
Media: ""application/json"" Data:
{
  "BadgeID": "{BadgeID GUID}",
  "Certification": {
    "CertificationID": "{CertID GUID}",
    "Description": "Certification Description", (not required for function)
    "ValidDays": 365, (not required for function – Short value from DB)
    "CertEnabled": 1 (not required for function – bool value from DB)
  },
  "CertDate": "{yyyy-mm-ddThh:mm:ss}",
```

//Above Values are required, below are optional

```
  "ExpDate": "{yyyy-mm-ddThh:mm:ss}",
  "Attempt": "1", (null, 1, 2, or 3)
  "Score": "100",
  "Result": "PASS", (null, PASS, or FAIL)
  "Translated": Flag, (null or 0 for off, 1 for on)
  "Proctor": "Text",
  "CertTaken": "Text",
  "Note": "Text",
  "RowID": "Text",
}
```

PUT /badges/certifications **Update a certification to an existing badge holder**

Parameters: {BadgeID} GUID must be defined in the media
 {Certification} GUID must be defined in the media
 Returns: OK

Example Method:

PUT |

Endpoint: `http://{Server}:{RestPort} |`
 Resource: `/pwapi/badges/certifications |`
 Media: `"application/json"` Data:

```
{
  "BadgeID": "{BadgeID GUID}",
  "Certification": {
    "CertificationID": "{CertID GUID}",
    "Description": "Certification Description", (not required for function)
    "ValidDays": 365, (not required for function – Short value from DB)
    "CertEnabled": 1 (not required for function – bool value from DB)
  },
  "CertDate": "{yyyy-mm-ddThh:mm:ss}",
  "ExpDate": "{yyyy-mm-ddThh:mm:ss}",
  "Attempt": "1", (null, 1, 2, or 3)
  "Score": "100",
  "Result": "PASS", (null, PASS, or FAIL)
  "Translated": Flag, (null or 0 for off, 1 for on)
  "Proctor": "Text",
  "CertTaken": "Text",
  "Note": "Text",
  "RowID": "Text",
}
```

GET /badges/{badge}/certifications

Get all certification for a badge holder

Parameters: {BadgeID} GUID

Returns: OK

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort} |`

Resource: `/pwapi/badges/{badge}/certifications`

Media: `"application/json"`

Data:

```
[
  {
    "BadgeID": "{BadgeID GUID}",
    "Certifications": {
      "CertificationID": "{Certificate ID}"
      "Description": "{Certificate Description}"
    }
    "CertDate": "{yyyy-mm-ddThh:mm:ss}",
    "ExpDate": "{yyyy-mm-ddThh:mm:ss}",
    "Attempt": "No of attempts",
    "Score": "Score obtained",
    "Result": "PASS/FAIL",
    "Translated": true/false,
    "Proctor": "string",
    "CertTaken": "String",
    "Note": "String",
    "RowID": "{ROWGUID}"
  },
  (Additional Certifications follow)
]
```

DELETE /badges/{badge}/certifications/{badgeID}/{rowID}**Delete a certification from a badge holder**

Parameters: {badge} & {BadgeID} GUID of badge holder
 {rowID} GUID of certification on badge holder (ROW_ID in PWNT DB)

Returns: OK

Example Method:

DELETE |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/certifications/{badgeID}/{rowID}|

GET /badges/blobs/paging/{pagesize}/{page}**GET a list of blobs by pagesize and page number**

Parameters: {pagesize} = Number of the quantity of badges per page
 {page} = Page number to request in the pages of results in quantity of {pagesize}

Returns: OK

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/paging/{pagesize}/{page}|

GET http://{server}:{port}/badges/blobs/paging/40/60

```
[
  {
    "BadgeID": "{BadgeID GUID}",
    "Blobs": [
      {
        "BlobID": null,
        "BlobType": {
          "BlobTypeID": "{BlobTypeID GUID}",
          "Description": "Badge Photo",
          "StoreFormat": 0,
          "FileFormat": 0,
          "BlobUser": null,
          "BlobPath": null,
          "BlobDate": "0001-01-01T00:00:00",
          "EDOCType": null,
          "LabelTag": null
        },
        "BlobValue": "/9j/--All Blob Data—" (Warning – Data can be large files)
      }
    ]
  }
]
(Additional Blobs follow for page quantity)
```

GET /badges/modified/{startdate}/{enddate}**GET a list of badgeholders with cards modified between the start and end dates provided.**

Parameters: {startdate} = Date in the format MM-DD-YYYY
 {enddate} = Date in the format MM-DD-YYYY

Returns: OK

ODATA Parameter: Query Last Name with:
 ?\$filter=LastName eq '{LNAME Text}'
 Only return data of First Name & Last Name with
 ?\$select=LastName, FirstName

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/modified/{startdate}/{enddate}|

GET http://{server}:{port}/badges/modified/10-01-2018/12-31-2018

RETURNS:

```
"BadgeID": Badge GUID,
"LastName": Text for last name,
"FirstName": Text for first name,
"MiddleName": Text for middle name,
"IssueDate": Badge Issue Date {yyyy-mm-ddThh:mm:ss},
"ExpireDate": Badge Expire Date {yyyy-mm-ddThh:mm:ss},
"BadgeType":
  "BadgeTypeID": BadgeType GUID,
  "Description": Badge Type Text

"BadgeStatus":
  "BadgeStatusID": BadgeStatus GUID,
  "Description": "Text Description" (active/inactive are default, all others custom)
--Then all custom BADGE_V fields are listed, as many as exist --
"CustomBadgeFields":
  "ColumnName": Text of Custom Column name,
  "FieldType": 0, (0 = DateTime SQL Type)
  "BooleanValue": null,
  "DateValue": {yyyy-mm-ddThh:mm:ss} (See GET /badge above for more examples)
--END Custom BADGE_V Field Section--
"Email": Text of Email field (can be Null),
"CellPhone": Text of Cell Phone field (can be Null),
"RowVersion": Hex GUID value for row version
"Cards":
  "BadgeID" = Badge GUID of Card Holder
  "CardNumber" = Number entered
  "CardStatus" = 0 (0, 1, 2, 3, ..., n) for reference values of status
  "PINCode" = {PIN Number}
  "IssueDate" = Card Issue Date {yyyy-mm-ddThh:mm:ss}
  "ExpireDate" = Card Expire Date {yyyy-mm-ddThh:mm:ss}
  "Company" = (group of company data for Card Company - 13 data fields)
  "Card Type":
    "BadgeTypeID" = BadgeType GUID
    "Description" = Description Text
    (And additional Card Fields, see GET badges/cards/{card})
--NEXT Badge modified between {startDate} and {EndDate} follows --
```

POST /badges/{badge}/partitions

Add a list of partitions to a badge holder

Parameters: {Badge} GUID of badge holder

Returns: 201 Created

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/partitions |

Media: ""application/json"" Data:

```
[
  {
```

```

    "PartitionID": "PartitionID GUID" (GUID starts 0x001B...)
  }, (Additional partitions follow as required)
]

```

DELETE /badges/{badge}/partitions/{partitionID}

Delete a partition from a badge holder

Parameters: {Badge} GUID of badge holder
 {partitionID} GUID of partition to be removed

Returns: OK

Example Method:

DELETE |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/{badge}/partitions/{partitionID} |

PUT /badges/terminate/{id}

Using the unique ID and the IDColumn from the config file, this function sets BADGE.BADGE_STATUS = 'Terminated', BADGE.EXPIRE_DATE = current date, terminates all active cards and sets BADGE_C.EXPIRE_DATE = current date

Parameters: Pro-Watch API Config file must be altered for method functionality.

```

<applicationSettings>
  <HoneywellAccess.ProWatch.PW_DTU_Service.Properties.Settings>
    <setting name="IDColumn" serializeAs="String">
      <value>BADGE_COLUMN_ID</value>
    </HoneywellAccess.ProWatch.PW_DTU_Service.Properties.Settings>
  </applicationSettings>

```

Set this value to the UNIQUE Badge column to use for badge/card terminations

{id} = the unique value in the BADGE_COLUMN_ID of the badge to be terminated.

Returns: OK

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/badges/terminate/{id} |

GET /badgestatuses

Get a list of all Badge Statuses. OData parameters are enabled

Returns: OK GET a list of Badgestatuses with GUID and description.

ODATA Parameter: Query Badge Status Name with:
 ?\$filter=Description eq '{Badgetype Text}'
 Only return data of Badge Status Description with
 ?\$select=Description

EXAMPLE: GET |

Endpoint: http://{Server}:{RestPort} |

Resource: pwapi/badgestatus

```

[
  {
    "BadgeStatusID": "{BadgeStatusID GUID}",
    "Description": "BadgeStatus Description Text"
  }
]

```

(Badgestatuses continue for all remaining types in database)

GET /badgetypes

Get a list of all Badge Types. OData parameters are enabled

Returns: OK GET a list of Badgetypes with GUID and description.

ODATA Parameter: Query BadgeType Name with:
 ?\$filter=Description eq '{Badgetype Text}'
 Only return data of Badgetype Description with
 ?\$select=Description

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/badgetype
 [
 {
 "BadgeTypeID": "{BadgetypeID GUID}",
 "Description": "Badgetype Description Text"
 }
] (Badgetypes continue for all remaining types in database)

GET /blobtypes

Get a list of all Blob Types. OData parameters are enabled

Returns: OK GET a list of Badgetypes with GUID and description.
 ODATA Parameter: Query BlobType Name with:
 ?\$filter=Description eq '{BlobType Text}'
 Only return data of Blobtype Description with
 ?\$select=Description

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/blobtype
 [
 {
 "BlobTypeID": "{BlobTypeID GUID}",
 "Description": "Blobtype Description Text"
 }
] (Blobtypes continue for all remaining types in database)

Delete /blobs

Delete blobs with the specified BlobID and BlobTypeID

Returns: OK
 EXAMPLE: DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: pwapi/blobs

[
 {
 "BlobID": "{Blob GUID}",
 "BlobTypeID": "{BlobTypeID GUID}",
 }
]

GET /certifications

Get all certifications. OData parameters are enabled

Returns: OK
 List of all Certifications in Pro-Watch.
 ODATA Parameter: Query Description with:
 ?\$filter= Description eq '{Description Text}'
 Return only Description results with
 ?\$select=Description

EXAMPLE: GET |
 Endpoint: http://{Server}:{RestPort} |

Resource: pwapi/certifications

```
[
  {
    "CertificationID": "CertificationID GUID",
    "Description": "Certification Description Text",
    "ValidDays": 365, (number of Certification Validity days from start date)
    "CertEnabled": true
  },
  {
    "CertificationID": "CertificationID GUID",
    "Description": "Certification Description Text",
    "ValidDays": 365, (number of Certification Validity days from start date)
    "CertEnabled": true
  }
] (List continues for all Certifications programmed in Pro-Watch)
```

GET /channels **Get all channels. OData parameters are enabled**

Returns: OK
List of all Channels in Pro-Watch with Site detail.
Similar result to /sites/{site}/channels, with all Pro-Watch channels listed

ODATA Parameter: Query Description with:
 ?\$filter= Description eq '{Description Text}'
 Return only Description results with
 ?\$select=Description

EXAMPLE: GET |

Endpoint: http://{Server}:{RestPort} |

Resource: pwapi/channels

```
[
  {
    "ChannelID": "PW-5000::0501", (Typical ChannelID for PW5000/6000/6101 Communications)
    "Description": "Channel Description",
    "Address": # of channel address at site,
    "ChannelType": 5, (# of all different PW Channel types)
    "Installed": true, (T/F)
    "Site": {
      "SiteID": "Site_text",
      "Description": "Site Description Text"
    }
  },
  {
    "ChannelID": "MMSystem::4201", (Typical ChannelID for MAXPRO VMS Communications)
    "Description": "MMSYSTEM", (MAXPRO VMS Channel does not change)
    "Address": 1,
    "ChannelType": 42, (Channel Type, MAXPRO VMS is type 42)
    "Installed": true,
    "Site": {
      "SiteID": "MMSystem",
      "Description": "VIDEO"
    }
  },
  {
    "ChannelID": "PW-5000::3903",
    "Description": "Vista Intrusion", (Channel Description Text, Vista Intrusion is type 39)
  }
]
```

```

    "Address": # of channel address at site,
    "ChannelType": 39,
    "Installed": false,
    "Site": {
        "SiteID": "Site_text",
        "Description": "Site Description Text"
    }
} (List continues for all Channels programmed in Pro-Watch)
]

```

GET /clearcodes **Get all clearance codes. OData parameters are enabled**

Returns: OK - All Clearance Codes in Pro-Watch with LogDevs & TimeZones detail.

ODATA Parameter: Query Clearance Code Name with:
 ?\$filter=Description eq '{Area Name Text}'
 Query multiple Clearance Code Names with "or":
 ?\$filter=Description eq 'A' or Description eq 'B'
 Only return data of Clearance Code Description with
 ?\$select=Description
 Only return data of Clearance Code GUID with
 ?\$select=ClearCodeID
 Return Both with ?\$select=ClearCodeID, Description

EXAMPLE: GET |

Endpoint: http://{Server}:{RestPort} |

Resource: pwapi/clearcodes

```

{
  "ClearCodeID": "ClearCodeID GUID",
  "Description": "ClearCode Description Text",
  "Description2": "",
  "Description3": "",
  "ClearCodeType": 0,
  "TimeZone": {
    "TimeZoneID": null,
    "Description": ""
  },
  "LogDevs": [
    {
      "ClearCodeID": "ClearCodeID GUID",
      "LogDev": {
        "LogDevID": "LogDevID GUID",
        "Description": "Logical Device Description Text"
      },
      "TimeZone": {
        "TimeZoneID": "TimeZoneID GUID",
        "Description": "Time Zone Description Text"
      }
    },
    {
      "ClearCodeID": "ClearCodeID GUID",
      "LogDev": {
        "LogDevID": "LogDevID GUID",
        "Description": "Logical Device Description Text"
      },
      "TimeZone": {
        "TimeZoneID": "TimeZoneID GUID",

```

```

        "Description": "Time Zone Description Text"
    }
} (List continues for all LogDevs programmed in this Clearance Code)
],
"ElevatorOutputs": []
} (List continues for all Clearance Codes programmed in Pro-Watch)

```

GET /clearcodes/{clearcode}/logicaldevs

Get a list of logical devices in a clearance code. OData parameters are enabled

Returns: OK – Single Clearance Code's LogicalDevs & TimeZones detail.

ODATA Parameter: Query Location with:
 ?\$filter=Location eq '{Location Text}'
 Return only Description results with
 ?\$select=Description
 Return a Nested Value like Logical Device Description with
 ?\$expand=LogDev(\$select=Description)

EXAMPLE: GET |

Endpoint: http://{Server}:{RestPort} |

Resource: pwapi/clearcodes/{clearcode}/logicaldevs

```

[
    {
        "ClearCodeID": "ClearCodeID GUID",
        "LogDev": {
            "LogDevID": "LogDevID GUID",
            "Description": "Logical Device Description Text"
        },
        "TimeZone": {
            "TimeZoneID": "TimeZoneID GUID",
            "Description": "Time Zone Description Text"
        }
    },
    {
        "ClearCodeID": "ClearCodeID GUID",
        "LogDev": {
            "LogDevID": "LogDevID GUID",
            "Description": "Logical Device Description Text"
        },
        "TimeZone": {
            "TimeZoneID": "TimeZoneID GUID",
            "Description": "Time Zone Description Text"
        }
    }
} (List continues for all LogDevs programmed in this Clearance Code)
]

```

GET /companies

Get all companies. OData parameters are enabled

Returns: OK

List of all Companies in PWNT with attributes.

ODATA Parameter: Query Description with:
 ?\$filter=Description eq '{Description Text}'
 Return only certain results with
 ?\$select=Description,Address1,City,State,Zip

EXAMPLE: GET |

Endpoint: `http://{Server}:{RestPort} |`

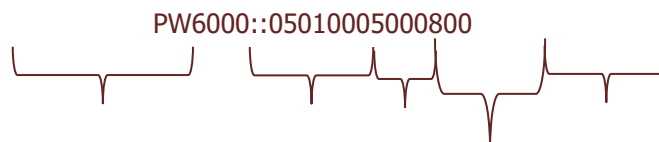
Resource: `pwapi/companies`

```
[
  {
    "ExpireMonths": 12,
    "Address1": Address1 Text,
    "Address2": Address2 Text,
    "City": City Text,
    "State": State Abbrev,
    "Zip": Zip Code Text,
    "CompanyTypeID": null,
    "FirmExpireDate": "YYYY-MM-DDTHH:MM:SS",
    "isExpireMethodInMonths": false,
    "CompanyID": "CompanyID GUID",
    "Description": "Company Description Text",
    "ClearCodes": null,
    "Partitions": null
  },
  {
    "ExpireMonths": 12,
    "Address1": null,
    "Address2": null,
    "City": null,
    "State": null,
    "Zip": null,
    "CompanyTypeID": null,
    "FirmExpireDate": "YYYY-MM-DDTHH:MM:SS",
    "isExpireMethodInMonths": false,
    "CompanyID": "CompanyID GUID",
    "Description": "Company Description Text",
    "ClearCodes": null,
    "Partitions": null
  }
] (List continues for all Companies programmed in Pro-Watch)
```

EVENTS

Events methods – will create an Event in the EV_Log for viewing in Event Monitor and Alarm Monitor. Events must contain ID, Date, and Type. Message and Card Number are optional, or if required for card events. Issuing events will contain the Pro-Watch Web API UserID for audit tracking.

Hardware IDs follow the Pro-Watch Naming Convention. Plain English names are used, with a series of numbers to identify the piece of hardware:



Site Name	Channel Panel	Logical Subpanel Device
-----------	---------------	----------------------------

POST /events/channel Issue a channel event

Parameters: {HardwareID}: Pro-Watch Hardware Channel ID NAME::NUMBER (eg: PW6000::0501)
 {EventDate}: DateTime of when this issued event happened
 {EventType}: Channel type of event (only channel events within this method.
 For a list of channel events, review the events tab in Channel Properties)
 {Message}: Text of message to accompany this event in Pro-Watch

Returns: 201 Created
 Location: EV_LOG

EXAMPLE:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /events/channel

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Communication Break" Channel Event ID=100 on a simulation channel)

http://server:port/pwapi/events/channel/

Application/Json Media:

```
{
  "HardwareID": "PW6000::0501",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 100,
  "Message": "Issued via Pro-Watch API"
}
```

POST /events/control Issue a control event

Parameters: {HardwareID}: Pro-Watch CTRL ID (eg: PASS3)
 {EventDate}: DateTime of when this issued event happened
 {EventType}: Control type of event (only control events within this method.
 For a list of control events, review the events tab in Server Options)
 {Message}: Text of message to accompany this event in Pro-Watch

Returns: 201 Created
 Location: EV_LOG

//If attempt has errors, returns 400 Bad Request, "Method can only issue an channel event type."

Current Pro-Watch Web API Supported Control Event Types:

```
EVLOG_THRESHOLD_LIMIT = 1,
Operator_LogOn=6,
Operator_Logoff=7,
CardValidationFailed = 41,
CardValidationPassed = 42,
DTU_MESSAGE = 43, // UNMATCHED MSG TYPE
PIVClassPAMMessage = 44, //pivCLASS Certificate Manager Activated
PIVClassCertMgrActivated = 45, //pivCLASS Certificate Manager Activated
PIVClassCertMgrRevoked = 46, //pivCLASS Certificate Manager Revoked
PIVClassCertMgrError = 47, //pivCLASS Certificate Manager Error
PIVClassValidationPassed = 48, //pivCLASS Passed Validation
PIVClassValidationFailed = 49, //pivCLASS Failed Validation
```

EXAMPLE:

POST |

Endpoint: `http://{Server}:{RestPort}` |Resource: `/events/control`

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Card Validation Failed" Panel Event ID=41 on a simulation control system)

`http://server:port/pwapi/events/input/`

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000500900",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 41,
  "Message": "Issued via Pro-Watch API",
  "CardNumber": "445566"
}
```

POST /events/input**Issue an input event**

Parameters: {HardwareID}: Pro-Watch Hardware Input ID NAME::NUMBER (eg: PW6000::0501000500900)
 {EventDate}: DateTime of when this issued event happened
 {EventType}: Input type of event (only input events within this method.
 For a list of input events, review the input events tab in Panel Properties)
 {Message}: Text of message to accompany this event in Pro-Watch

Returns: 201 Created
 Location: EV_LOG

EXAMPLE:

POST |

Endpoint: `http://{Server}:{RestPort}` |Resource: `/events/input`

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Input Point in Trouble" Panel Event ID=904 on a simulation panel)

`http://server:port/pwapi/events/input/`

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000500900",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 904,
  "Message": "Issued via Pro-Watch API"
}
```

("Input Point in Trouble" Input Event ID=904 can have a "Return to Normal" by issuing "EventType": 10904)

`http://server:port/pwapi/events/input/`

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000500900",
```

```
"EventDate": "2018-11-19T10:12:45",
"EventType": 10904,
"Message": "Issued via Pro-Watch API"
```

```
}
```

(Pro-Watch Event Monitor Example)

Event Category	Resource Id	Event Time	Logical Device Description	Point Description	Event Type Description	Message
Event Occurred	PW-5000::05010005000900	11/19/2018 1:12:45 PM	Front Door Entrance	Input point in trouble RTN	Input point fault detected	Issued via Pro-Watch API
Event Occurred	PW-5000::05010005000900	11/19/2018 1:12:45 PM	Front Door Entrance	Input point in trouble	Input point fault detected	Issued via Pro-Watch API

POST /events/output**Issue an output event**

Parameters: {HardwareID}: Pro-Watch Hardware Output ID NAME::NUMBER (eg: PW6000::05010005001000)

{EventDate}: DateTime of when this issued event happened

{EventType}: Output type of event (only output events within this method.

For a list of output events, review the output events tab in Panel Properties)

{Message}: Text of message to accompany this event in Pro-Watch

Returns: 201 Created

Location: EV_LOG

EXAMPLE:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /events/output

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Output Point is Active" Panel Event ID=950 on a simulation panel)

http://server:port/pwapi/events/output/

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000501000",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 950,
  "Message": "Issued via Pro-Watch API"
}
```

("Output Point is Active" Output Event ID=950 can have a "Return to Normal" by issuing "EventType": 10950)

http://server:port/pwapi/events/output/

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000501000",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 10950,
  "Message": "Issued via Pro-Watch API"
}
```

(Pro-Watch Event Monitor Example)

Event Category	Resource Id	Event Time	Logical Device Description	Point Description	Event Type Description	Message
Event Occurred	PW-5000::05010005001000	11/19/2018 1:12:45 PM	Front Door Entrance	Output point is active RTN	Output Active	Issued via Pro-Watch API
Event Occurred	PW-5000::05010005001000	11/19/2018 1:12:45 PM	Front Door Entrance	Output point is active	Output Active	Issued via Pro-Watch API

POST /events/panel**Issue a panel event**

Parameters: {HardwareID}: Pro-Watch Hardware Panel ID NAME::NUMBER (eg: PW6000::050101)

{EventDate}: DateTime of when this issued event happened

{EventType}: Panel type of event (only panel events within this method.

For a list of panel events, review the events tab in Panel Properties)

Returns: {Message}: Text of message to accompany this event in Pro-Watch
201 Created
Location: EV_LOG

EXAMPLE:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: /events/panel

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Panel Tamper" Panel Event ID=202 on a simulation panel)

<http://server:port/pwapi/events/panel/>

Application/Json Media:

```
{
  "HardwareID": "PW-6000::050101",
  "EventDate": "2018-11-19T12:02:45",
  "EventType": 202,
  "Message": "Issued via Pro-Watch API"
}
```

("Panel Tamper" Panel Event ID=202 can have a "Return to Normal" by issuing "EventType": 10202)

<http://server:port/pwapi/events/panel/>

Application/Json Media:

```
{
  "HardwareID": "PW-6000::050101",
  "EventDate": "2018-11-19T12:02:45",
  "EventType": 10202,
  "Message": "Issued via Pro-Watch API"
}
```

(Pro-Watch Alarm Monitor Example)

Disposition	Rollup #	Company	Last Name	First Name	Event Type Descriptio	Logical Device Descript	System Time	Event Time	Message	Logical Devi
RCV	1				Panel Tamper Alarm	PW-2000 Panel	11/19/2018 12:06:41 PM	11/19/2018 12:02:45 PM	Issued via Pro-Watch API	
RCV	1				Panel Tamper Alarm	PW-2000 Panel	11/19/2018 12:05:27 PM	11/19/2018 12:02:45 PM	Issued via Pro-Watch API	

POST /events/reader

Issue a reader event

READER Events can contain Card Numbers – for card Activity events (local grant, unknown card). They may not have to contain a card number (reader in card-only mode). If a card number is POSTed to the reader, the event will also contain the Badge holder information associated to that card, if such a badge exists.

Parameters: {HardwareID}: Pro-Watch Hardware Panel ID NAME::NUMBER (eg: PW6000::05010005000800)
{EventDate}: DateTime of when this issued event happened
{EventType}: Reader type of event (only Reader events within this method.

For a list of Reader events, review the events tab in Reader Properties)

Returns: {Message}: Text of message to accompany this event in Pro-Watch
201 Created
Location: EV_LOG

EXAMPLE:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: /events/reader

```
{
  "HardwareID": "NAME::NUMBER",
```

```
"EventDate": "YYYY-MM-DDTHH:MM:SS",
"EventType": EventNumber,
"Message": "Message Text",
"CardNumber": "Card Number"
```

```
}
```

(Real Example for issuing a "Local Grant" Reader Event ID=500 on a simulation panel)

<http://server:port/pwapi/events/subpanel/>

Application/Json Media:

```
{
  "HardwareID": "PW-6000::05010005000800",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 500,
  "Message": "Issued via Pro-Watch API",
  "CardNumber": "445566"
}
```

(After several different event POST results: Pro-Watch Event Monitor Example)

Event Category	Resource Id	Event Time	Logical Device Description	Point Description	Event Type Description	Message	Card Number
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Host Flex, Door not used	Request to Exit	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Reader has been locked	Reader Mode Locked	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Executive Privilege	Executive Privilege	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Host Grant	Host Grant	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Local Grant	Local Grant	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Coax Failure	Subpanel Down	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Valid card at an unauthorized reader	Invalid Reader	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Card Trace	Badge Trace	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Expired Card Attempt	Expired Badge	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Void Card	Void Card	Issued via Pro-Watch API	445566
Event Occurred	PW-5000::05010005000800	11/19/2018 2:02:45 PM	Front Door Entrance	Unknown Card	Unknown Badge	Issued via Pro-Watch API	445566

POST /events/subpanel Issue a subpanel event

Parameters: {HardwareID}: Pro-Watch Hardware Subpanel ID NAME::NUMBER (eg: PW6000::0501000500)

{EventDate}: DateTime of when this issued event happened

{EventType}: Panel type of event (only subpanel events within this method.
For a list of subpanel events, review the events tab in Panel Properties)

{Message}: Text of message to accompany this event in Pro-Watch

Returns: 201 Created

Location: EV_LOG

EXAMPLE:

POST |

Endpoint: <http://{Server}:{RestPort}/events/subpanel/>

Resource: /events/subpanel

```
{
  "HardwareID": "NAME::NUMBER",
  "EventDate": "YYYY-MM-DDTHH:MM:SS",
  "EventType": EventNumber,
  "Message": "Message Text"
}
```

(Real Example for issuing a "Subpanel Comms Disabled" Panel Event ID=304 on a simulation panel)

<http://server:port/pwapi/events/subpanel/>

Application/Json Media:

```
{
  "HardwareID": "PW-6000::0501000500",
  "EventDate": "2018-11-19T10:12:45",
  "EventType": 304,
  "Message": "Issued via Pro-Watch API"
}
```

GET /hwclass/{hwclass}/logdevs

Get all logical devices in a hardware class.

Get all logical devices in a hardware class.

Parameters: {hwclass} = HardwareClass GUID from HW_CLASS table in PWNT.
 Returns: OK
 Return Data dependent upon which Class type is chosen. Can be null.

ODATA Parameter: Query Location with:
 ?\$filter=Location eq '{Location Text}'
 Return only Description results with
 ?\$select=Description

EXAMPLE: (Similar Structure to LogDevs with the addition of "Panel" result)

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /hwclass/{hwclass}/logdevs

```
[
  {
    "LogDevID": "0x006F...", (LOGDEV GUID)
    "Description": "Logical Device Description Text",
    "AltDescription": "Alternate Description Text",
    "Location": "Location Description Text",
    "HWTemplate": {
      "HWTemplateID": "Hardware Template GUID",
      "Description": "Hardware Description Text"
    },
    "Panel": {
      "PanelID": "PANEL_ID", (HEXDEC Text: e.g. PW6000::050100)
      "Description": "Panel Description Text",
      "Installed": true
    },
    "HWClass": {
      "HWClassID": "Hardware Class GUID",
      "Description": "Hardware Class Description Text"
    },
    "Site": {
      "SiteID": "SiteID TEXT",
      "Description": "Site Description Text"
    }
  },
  (2nd Device Follows, and so-on for all Logical Devices in this class)
]
```

GET /logdevs

Get all logical devices. OData parameters are enabled

Parameters: OPTIONAL /logdevs/{hardware=false} TRUE/FALSE option to show all LogDev hardware.
 Returns: OK All Logical Device Data includes Class and Site.

ODATA Parameter: Query Location with:
 ?\$filter=Location eq '{Location Text}'
 Return only Description results with
 ?\$select=Description

EXAMPLE:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/logdevs

```
[
  {
    "LogDevID": "0x006F...", (LOGDEV GUID)
    "Description": "Logical Device Description Text",
    "AltDescription": "Alternate Description Text",
    "Location": "Location Description Text",
  }
]
```

```

    "HWTemplate": {
      "HWTemplateID": "Hardware Template GUID",
      "Description": "Hardware Description Text"
    },
    "HWClass": {
      "HWClassID": "Hardware Class GUID",
      "Description": "Hardware Class Description Text"
    },
    "Site": {
      "SiteID": "SiteID TEXT",
      "Description": "Site Description Text"
    }
  },
  (2nd Device Follows, and so-on for all Logical Devices)
  {
    "LogDevID": "0x006F...", (LOGDEV GUID)
    "Description": "Logical Device Description Text",
    "AltDescription": "Alternate Description Text",
    "Location": "Location Description Text",
    "HWTemplate": {
      "HWTemplateID": "Hardware Template GUID",
      "Description": "Hardware Description Text"
    },
    "HWClass": {
      "HWClassID": "Hardware Class GUID",
      "Description": "Hardware Class Description Text"
    },
    "Site": {
      "SiteID": "V3-PW6101",
      "Description": "V3-PW6101 Demo Case"
    }
  }
]

```

GET /logdevs/{logdev}/hardware Get the hardware for a logical device. OData parameters are enabled

Parameters: {logdev} = LogicalDeviceID GUID

Returns: OK and all Hardware Data

ODATA Parameter: Query Hardware Type with:

?\$filter=DeviceTypeDesc eq 'Reader' (only returns Reader information)

Return only Description Results by "HardwareDesc" with

?\$select=HardwareDesc

Example: GET hardware for a Pro-Watch Door Typical ACR (Reader) – default template has 4 hardware items

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/logdevs/{logdev}/hardware

```

[
  {
    "LogDevID": "LogicalDeviceID GUID",
    "HardwareID": "SITE::HARDWAREID", [AlphaNumeric PWNT Value] (example: PW-6101::05010005000800)
    "HardwareDesc": "Hardware Description – Hardware Device" e.g.: "Reader"
    "DeviceTypeID": "DeviceType GUID",
    "DeviceTypeDesc": "Device Type Description", (Reader|Door Position|RexDevice|Lock|etc.)
    "HardwareType": 8, (Numeric Value for Hardware Type)
  }
]

```



```

    "Installed": true
  },
  {
    "LogDevID": "LogicalDeviceID GUID",
    "HardwareID": "SITE::HARDWAREID", [AlphaNumeric PWNT Value] (example: PW-6101::05010005000900)
    "HardwareDesc": "Hardware Description – Hardware Device", e.g.: "Door Position"
    "DeviceTypeID": "DeviceType GUID",
    "DeviceTypeDesc": "Device Type Description", (Reader|Door Position|RexDevice|Lock|etc.)
    "HardwareType": 9, (Numeric Value for Hardware Type)
    "Installed": true
  },
  {
    "LogDevID": "LogicalDeviceID GUID",
    "HardwareID": "SITE::HARDWAREID", [AlphaNumeric PWNT Value] (example: PW-6101::05010005000901)
    "HardwareDesc": "Hardware Description – Hardware Device", e.g.: "Rex Device"
    "DeviceTypeID": "DeviceType GUID",
    "DeviceTypeDesc": "Device Type Description", (Reader|Door Position|RexDevice|Lock|etc.)
    "HardwareType": 9, (Numeric Value for Hardware Type)
    "Installed": true
  },
  {
    "LogDevID": "LogicalDeviceID GUID",
    "HardwareID": "SITE::HARDWAREID", [AlphaNumeric PWNT Value] (example: PW-6101::05010005001000)
    "HardwareDesc": "Hardware Description – Hardware Device", e.g.: "Lock"
    "DeviceTypeID": "DeviceType GUID",
    "DeviceTypeDesc": "Device Type Description", (Reader|Door Position|RexDevice|Lock|etc.)
    "HardwareType": 10, (Numeric Value for Hardware Type)
    "Installed": true
  }
]

```

GET /logdevs/{logdev}/{card}/{datetime}/{unlock=false} **Gets status of card access at logical device for specified time.**

Parameters: {logdev} = LogicalDeviceID GUID
 {CardNo} = Card Number from PW Database
 {CurrDateTime} = Date & Time with format YYYY-MM-DDThh:mm:ss
 {unlock} = Boolean to control issue of momentary unlock door

Returns: OK, BAD, Unauthorized depending on card access to device

EXAMPLE: GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `pwapi/areas/{area}/{CardNo}/{CurrDateTime}`

Response 200 OK – if CardNo Exists and issues momentary unlock when required

Response 400 BAD REQUEST "Card number not found in Pro-Watch database [CardNo]" – if CardNo does not exist

Response 401 Unauthorized – if CardNo Exists but card is not allowed in area at request DateTime

DOES HAVE ACCESS (LogDev) By CardNumber and Current DateTime

POST /logdevs/{logdev}/lock **Lock a logical device/door**

Parameters: {logdev} = LOGICAL_DEV ID GUID for a READER (momentary unlock command posts to reader)
 {logdev} GUID should start as 0x006F...

Returns: OK
 Pro-Watch Event Log will show:
 Reader/Door Locked: Door Locked

Event Occurred: "SITE::HARDWAREID" – "Reader has been Locked"
Valid card reads as "Attempt to open Locked Door"

Example Method:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/logdevs/{logdev}/lock> |

POST /logdevs/{logdev}/momentaryunlock **Momentary unlock a logical device/door**

Parameters: {logdev} = LOGICAL_DEV ID GUID for a READER (lock command posts to reader)
 {logdev} GUID should start as 0x006F...

Returns: OK

Pro-Watch Event issued: Host Rex, Non-Verified. Hardware Transaction Report lists Request-to-Exit

Pro-Watch Event Log will show:

Momentary Unlock: Momentary unlock

Event Occurred: Host Rex, Non-Verified

Valid card reads as "Open Unlocked Door" if within the momentary shunt time

Example Method:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/logdevs/{logdev}/momentaryunlock> |

POST /logdevs/{logdev}/timeoverride/{second} **Time override a door**

Parameters: {logdev} = LOGICAL_DEV ID GUID for a READER (unlock command posts to reader)
 {logdev} GUID should start as 0x006F...

{second} = Number of seconds to hold open that door (numeric value)

Returns: OK

Pro-Watch Event Log will show:

Reader/Door Timed Override: Timed override issue

Event Occurred: Host Rex, Non-Verified

Event Occurred: Timed Override enabled

{Duration occurs for the number of seconds provided in the POST command}

Event Occurred: Timed Override disabled

Example Method:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/logdevs/{logdev}/timeoverride/{second}> |

<http://{Server}:{RestPort}/pwapi/logdevs/0x006F.../timeoverride/6> (this issues a 6 second timed override)

POST /logdevs/{logdev}/unlock **Unlock a logical device/door**

Parameters: {logdev} = LOGICAL_DEV ID GUID for a READER (unlock command posts to reader)
 {logdev} GUID should start as 0x006F...

Returns: OK

Pro-Watch Event Log will show:

Reader/Door Unlocked: Door unlocked

Event Occurred: "SITE::HARDWAREID" – "Reader has been unlocked"

Valid card reads as "Open Unlocked Door"

Example Method:

POST |

Endpoint: <http://{Server}:{RestPort}> |

Resource: </pwapi/logdevs/{logdev}/unlock> |

POST /logdevs/{logdev}/reenable TZ**Return a logical device/door to its normal operation for**

Parameters: {logdev} = LOGICAL_DEV ID GUID for a READER (Re-Enable command posts to reader)
 {logdev} GUID should start as 0x006F...

RE-ENABLE command is used to return a reader to its normal state following a "Lock" command

Returns: OK

Pro-Watch Event Log will show:

Reader/Door Access: Door Reenabled

Event Occurred: "SITE::HARDWAREID" – "Reader in Card Only Mode" (mode of TZ)

Valid card reads as "Local Grant"

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/logdevs/{logdev}/reenable |

GET /panels**Gets all panels. OData parameters are enabled**

Parameters: {none}

ODATA Parameter: Query Panel Description with:
 ?\$filter=Description eq '{Panel Description Text}'
 Order Panel Results by "Location" with
 ?\$orderby=Location

Returns: OK

Data returns in same format as GET /sites/{site}/panels – but returns all panels in Pro-Watch, from all sites.

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/panels |

```
[
  {
    "PanelID": "PW-5000::050100", (Typical PanelID for PW6000 Panel Programming)
    "Description": "Panel Description Text",
    "Location": "Panel Location Text",
    "Installed": true, (T/F)
    "Channel": {
      "ChannelID": "ChannelID_Text",
      "Description": "Channel Description",
      "Installed": true
    },
    "Site": {
      "SiteID": " Site_text",
      "Description": "Site Description Text"
    }
  },
  {
    "PanelID": "PanelID::TEXT",
    "Description": "Panel Description Text",
    "Location": "Location Description Text",
    "Address": 1,
    "Installed": true,
    "Channel": {
      "ChannelID": "ChannelID Text",
      "Description": "Channel Description Text",

```

```

    "Installed": false
  },
  "Site": {
    "SiteID": "Site_text",
    "Description": "Site Description Text"
  }
}

```

] (Panel responses continue for as many Panels as exists in the System)

GET /panels/{panel}/timezones Get all time zones for a panel. OData parameters are enabled

Parameters: {Panel} = Panel ID Text (example PW-6000::050100)

Returns: OK

Lists all Time Zones programmed to that panel.

ODATA Parameter: Query TimeZone Description with:
 ?\$filter=Description eq '{TimeZone Description Text}'
 Return only Description fields with:
 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/panels/{PanelID:TEXT}/timezones |

```

[
  {
    "TimeZoneID": "TimeZoneID GUID",
    "Description": "System All Times" (Time Zone Description Text)
  },
  {
    "TimeZoneID": "TimeZoneID GUID",
    "Description": "Time Zone Description Text"
  }
] (List continues for remaining panel time zones)

```

GET /partitions Get partitions. OData parameters are enabled

Returns: OK

Lists the partitions programmed in the Pro-Watch system.

ODATA Parameter: Query Partition Description with:
 ?\$filter=Description eq '{Partition Description Text}'
 Return only Description fields with:
 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/partitions |

```

[
  {
    "PartitionID": "PartitionID GUID", (GUID starts 0x001B...)
    "Description": "Default", (Partition Description Text)
    "isSelected": false,
    "FieldValue": null,
    "ResourceType": 0
  },
  {
    "PartitionID": "PartitionID GUID", (GUID starts 0x001B...)
  }
]

```

```

    "Description": "Partition Description Text",
    "isSelected": false,
    "FieldValue": null,
    "ResourceType": 0
  } (List continues for remaining system partitions)
]

```

GET /profiles/{badge_profID}

Get information on a Badge Profile.
OData parameters are enabled

Parameters: {badge_profID}: "BadgeProfileID GUID" GUID of the Badge Profile.

Returns: OK

Lists the pages and fields programmed in the Pro-Watch system.

ODATA Parameter: Query Description with:

?\$filter=Description eq '{Badge Profile name Text}'

Return only Description fields with:

?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/profiles/{badge_profID} |

```

{
  "Pages": [
    {
      "Fields": [
        {
          "ProfileID": "{badge_profIDGUID}",
          "PageID": "{pageIDGUID}",
          "ColumnName": "BADGE_COLUMN_NAME",
          "PosX": 353, (pixel values of where this field appears in the badging GUI)
          "PosY": 21,
          "PosWidth": 205,
          "PosHeight": 237
        },
        (additional fields follow for all of the fields on this badge page)
      ],
      "ProfileID": "{badge_profIDGUID}",
      "PageID": "{pageIDGUID}",
      "Description": "Page Description (badging tab)",
      "PageOrder": 1
    },
    {
      "Fields": [
        {
          "ProfileID": "{badge_profIDGUID}",
          "PageID": "{pageIDGUID}",
          "ColumnName": "BADGE_COLUMN_NAME",
          "PosX": 353,
          "PosY": 21,
          "PosWidth": 205,
          "PosHeight": 237
        },
        {
          "ProfileID": "{badge_profIDGUID}",

```

```

    "PageID": "{pageIDGUID}",
    "Description": "Page Description (badging tab)",
    "PageOrder": 2
  },
],      (additional Pages follow for all of the tabs on this badge profile)
"ProfileID": "{badge_profIDGUID}",
"Description": "Badge Profile Description",
"CardsTab": true      (true/false – whether the "Cards Tab" is visible in this profile)
}

```

GET /programs**Get program and function access in Pro-Watch**

Returns: OK

Defines the program functionality for requests:

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/programs |

```

{
  "Badging": {
    "Query": true,
    "Add": true,
    "Update": true,
    "Delete": true
  },
  "Site": {
    "Query": true,
    "Add": true,
    "Update": true,
    "Delete": true
  }
}

```

GET /sites**Get sites. OData parameters are enabled**

Returns: OK

ODATA Parameter: Query Sites with:
 ?\$filter=SiteID eq '{Site name text}'
 Return only Description fields with:
 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/sites |

```

[
  {
    "SiteID": "Site_text",
    "Description": "Site Description Text"
  },
  {
    "SiteID": "Site2_text",
    "Description": "Site2 Description Text"
  }
] (Site responses continue for as many sites as exists in the DB)

```

GET /sites/{site}/channels**Get all channels for a site. OData parameters are enabled**

Parameters: {site} = "Site_text"

Returns: OK

ODATA Parameter: Query Channels with:
 ?\$filter=ChannelID eq '{ChannelID::TEXT}'
 Return only Description fields with:
 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/sites/{site}/channels |

[Example contains a response with 2 channels programmed the {site}]

```
[
  {
    "ChannelID": "PW-5000::0501",      (Typical ChannelID for PW5000/6000/6101 Communications)
    "Description": "Channel Description",
    "Address": # of channel address at site,
    "ChannelType": 5, (# of all different PW Channel types)
    "Installed": true, (T/F)
    "Site": {
      "SiteID": "Site_text",
      "Description": "Site Description Text"
    }
  },
  {
    "ChannelID": "PW-5000::3903",
    "Description": "Vista Intrusion", (Channel Description Text, Vista Intrusion is type 39)
    "Address": # of channel address at site,
    "ChannelType": 39,
    "Installed": false,
    "Site": {
      "SiteID": "Site_text",
      "Description": "Site Description Text"
    }
  }
] (Channel responses continue for as many Channels as exists in the Site)
```

GET /sites/{site}/logdevs
enabled**Get all logical devices for a site. OData parameters are**

Parameters: {site} = "Site_text"

Returns: OK

ODATA Parameter: Query LogDevs with:
 ?\$filter=Description eq '{LogDev description text}'
 Return Description & Location fields with:
 ?\$select=Description,Location

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/sites/{site}/logdevs |

[Example contains a single Logical Device response]

```
{
  "LogDevID": "Logical Device ID GUID",
```

```

"Description": "Device Description text",
"AltDescription": "Alternate Description text",
"Location": "Location text",
"HWTemplate": {
  "HWTemplateID": "Hardware Template GUID",
  "Description": "Hardware Template Description text"
},
"HWClass": {
  "HWClassID": "Hardware Class GUID",
  "Description": "Hardware Class Description text"
},
"Site": {
  "SiteID": "Site_text",
  "Description": "Site Description Text"
}
} (Logical Device responses continue for as many LogDevs as exists in the Site)

```

GET /sites/{site}/panels Get all panels for a site. OData parameters are enabled

Parameters: {site} = "Site_text"

Data returns in same format as GET /panels – but returns panels in specified site.

Returns: OK

ODATA Parameter: Query Panels with:
 ?\$filter=Description eq '{Panel description text}'
 Return Description & Location fields with:
 ?\$select=Description,Location

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/sites/{site}/panels |

[Example contains a response with 2 Panels programmed at the {site}]

```

[
  {
    "PanelID": "PW-5000::050100", (Typical PanelID for PW6000 Panel Programming)
    "Description": "Panel Description Text",
    "Location": "Panel Location Text",
    "Installed": true, (T/F)
    "Channel": {
      "ChannelID": "ChannelID_Text",
      "Description": "Channel Description",
      "Installed": true
    },
    "Site": {
      "SiteID": " Site_text",
      "Description": "Site Description Text"
    }
  },
  {
    "PanelID": "PanelID_TEXT",
    "Description": "Panel Description Text",
    "Location": "Location Description Text",
    "Address": 1,
    "Installed": true,
    "Channel": {

```



```

    "ChannelID": "ChannelID Text",
    "Description": "Channel Description Text",
    "Installed": false
  },
  "Site": {
    "SiteID": "Site_text",
    "Description": "Site Description Text"
  }
}

```

] (Panel responses continue for as many Panels as exists in the Site)

GET /sites/{site}/subpanels Get all subpanels for a site. OData parameters are enabled

Parameters: {site} = "Site_text"

Data returns in same format as GET /subpanels – but returns subpanels in specified site.

Returns: OK

ODATA Parameter: Query Panels with:
 ?\$filter=Description eq '{Subpanel description text}'
 Return Description & Location fields with:
 ?\$select=Description,Location

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/sites/{site}/subpanels |

[Example contains a response with 1 Subpanel programmed at the {site}]

```

[
  {
    "SubpanelID": "PW-5000::0501000500", (Typical SubpanelID for PW6K1R2 subpanel Programming)
    "Description": "Panel Description Text",
    "Location": "Location Description Text",
    "Installed": true,
    "Panel": {
      "PanelID": "PW-5000::050100", (Typical PanelID for PW6000 Panel Programming)
      "Description": "Panel Description Text",
      "Channel": {
        "ChannelID": "ChannelID Text",
        "Description": "Channel Description Text",
      },
      "Site": {
        "SiteID": "Site_text",
        "Description": "Site Description Text"
      }
    }
  }
]

```

] (Subpanel responses continue for as many Subpanels as exists in the Site)

GET /sites/{site}/summary Get a site count summary of items programmed to that {site}. OData Parameters are enabled.

Parameters: {site} = "Site_text"

Returns: OK

ODATA Parameter: Return Description & Count fields with:
 ?\$select=ItemDesc,ItemCount

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort} |`

Resource: `/pwapi/sites/{site}/summary |`

[Example contains a response with 1 Panel and 3 Device Types programmed the {site}]

```
[
  {
    "ItemID": "",
    "ItemDesc": "Channels",
    "ItemCount": 1, (Number of channels that exist in this site)
    "PWResource": 3
  },
  {
    "ItemID": "",
    "ItemDesc": "Panels",
    "ItemCount": 1, (Number of panels that exist in this site)
    "PWResource": 4
  },
  {
    "ItemID": "{HW_CLASS GUID}",
    "ItemDesc": "Readers",
    "ItemCount": 2, (Number of readers that exist in this site)
    "PWResource": 111
  },
  {
    "ItemID": "{HW_CLASS GUID}",
    "ItemDesc": "Monitorable Inputs",
    "ItemCount": 4, (Number of inputs that exist in this site)
    "PWResource": 111
  },
  {
    "ItemID": "{HW_CLASS GUID}",
    "ItemDesc": "Controllable Outputs",
    "ItemCount": 4, (Number of outputs that exist in this site)
    "PWResource": 111
  }
]
```

GET /subpanels

Gets all subpanels. OData parameters are enabled

ODATA Parameter: Query Subpanel Description with:
 ?\$filter=Description eq '{Panel Description Text}'
 Order Subpanel Results by "Location" with
 ?orderby=Location
 Return only certain fields with:
 ?select=SubpanelID,Description

Returns: OK

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort} |`

Resource: `/pwapi/subpanels |`

```
[
  {
    "SubpanelID": "SubpanelID::TEXT",
    "Description": "SubpanelType Description 0", (0= Subpanel address on IC Panel)
```

```

"Location": "Location Text",
"Installed": true,
"Panel": {
  "PanelID": "PanelID::TEXT",
  "Description": "IC Panel Description",
  "Installed": false,
  "Channel": {
    "ChannelID": "ChannelID::TEXT",
    "Description": "Channel Description",
    "Installed": false
  },
  "Site": {
    "SiteID": "SiteName",
    "Description": "SiteDescription"
  }
}
}
]

```

(can be True/False)

(Results continue for all additional system Subpanels)

GET /timezones**Get timezones. OData parameters are enabled**

Returns: OK (only returns Day values if that day is present in TZ)

ODATA Parameter: Query timezones with:

 ?\$filter=Description eq '{Description text}'

 Return only Description fields with:

 ?\$select=Description

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/timezones |

```

[
  {
    "TimeZoneID": "TimeZone GUID",
    "Description": "System All Times",
    "Schedules": [
      {
        "TimeZoneID": "TimeZone GUID",
        "StartTime": "00:00:00",
        "StopTime": "23:59:00",
        "InUse": true,
        "Monday": true,
        "Tuesday": true,
        "Wednesday": true,
        "Thursday": true,
        "Friday": true,
        "Saturday": true,
        "Sunday": true,
        "Holiday1": true,
        "Holiday2": true,
        "Holiday3": true
      }
    ]
  }
]

```

] (Next Example: Gapped Time Zone where certain days use differing times)

```
{
  "TimeZoneID": "TimeZone GUID",
  "Description": "MTWTF 09:00-17:00 & Sat 09:00-13:00",
  "Schedules": [
    {
      "TimeZoneID": "TimeZone GUID",
      "StartTime": "09:00:00",
      "StopTime": "17:00:00",
      "InUse": true,
      "Monday": true,
      "Tuesday": true,
      "Wednesday": true,
      "Thursday": true,
      "Friday": true
    },
    {
      "TimeZoneID": "TimeZone GUID",
      "StartTime": "09:00:00",
      "StopTime": "13:00:00",
      "InUse": true,
      "Saturday": true,
      "Sunday": true
    }
  ]
}
} (TimeZone responses continue for as many TimeZones as exists in the DB)
```

POST /timezones**Add timezone and schedules to PW database**

Returns: OK (201) if adding timezone was successful

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/timezones |

Media: "Application/Json:"

Data:

```
{
  "Description": "New Timezone Description",
  "Schedules": [
    {
      "StartTime": "HH:MM:00", (Time in 24 HR format)
      "StopTime": "HH:MM:00", (Time in 24 HR format)
      "InUse": true/false,
      "Monday": true/false,
      "Tuesday": true/false,
      "Wednesday": true/false,
      "Thursday": true/false,
      "Friday": true/false,
      "Saturday": true/false,
      "Sunday": true/false,
      "Holiday1": true/false,
      "Holiday2": true/false,
      "Holiday3": true/false
    },
  ],
}
```

(As many schedules as you required to be added for the TimeZone)

```
]
}
```

PUT /timezones/{timezoneID} Updates an existing timezone and issues a timezone download command to affected panels.

Parameters: {timezoneID} = "Timezone GUID"

Returns: OK (200) if updating timezone was successful

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/timezones/{timezoneID} |

Media: "Application/Json"

Data:

```
{
  "TimeZoneID": " timezoneID",
  "Description": "Updated Timezone Description",
  "Schedules": [
    {
      "StartTime": "HH:MM:00", (Time in 24 HR format)
      "StopTime": "HH:MM:00", (Time in 24 HR format)
      "InUse": true/false,
      "Monday": true/false,
      "Tuesday": true/false,
      "Wednesday": true/false,
      "Thursday": true/false,
      "Friday": true/false,
      "Saturday": true/false,
      "Sunday": true/false,
      "Holiday1": true/false,
      "Holiday2": true/false,
      "Holiday3": true/false
    },
    (As many schedules as you required to be updated for the TimeZone)
  ]
}
```

GET /users Gets all users. OData parameters are enabled

Returns: OK (returns the list of users)

ODATA Parameter: Query users with:
 ?\$filter=FirstName eq '{FirstName Text}'
 Return only UserName fields with:
 ?\$select=UserName

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/users |

```
[
  {
    "UserID": "{UserID}",
    "Description": "Users Description",
```

```

    "EMail": "Users Email",
    "FirstName": "Users FirstName",
    "LastName": "Users LastName",
    "UserName": "Users username",
    "Expires": "Expiration date of the user",
    "ClassID": "{ClassID} of the users Class",
    "CellPhone": "Users Cell Phone"
  }
]

```

POST **/users** **Adds a user using PwUser object**

Returns: OK

Returns the UserID {ItemID} of the newly added record

Note: Password is not implemented and must be done via the Badging User Interface

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/users |

```

[
  {
    "Description": "Users Description",
    "EMail": "Users Email",
    "FirstName": "Users FirstName",
    "LastName": "Users LastName",
    "UserName": "Users username",
    "Expires": "Expiration date of the user",
    "ClassID": "{ClassID} of the users Class",
    "CellPhone": "Users Cell Phone"
  }
]

```

PUT **/users** **Updates a user using PwUser object**

Returns: OK

Note: Password is not implemented and must be done via the Badging User Interface

Example Method:

PUT |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/users |

```

[
  {
    "UserID": "{UserID}",
    "Description": "Users Description",
    "EMail": "Users Email",
    "FirstName": "Users FirstName",
    "LastName": "Users LastName",
    "UserName": "Users username",
    "Expires": "Expiration date of the user",
    "ClassID": "{ClassID} of the users Class",
    "CellPhone": "Users Cell Phone"
  }
]

```

]

DELETE /users/{userID} DeSeletes a user with the specified user ID

Parameters: {userID} = UserID GUID
 Returns: OK

Method: DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID} |

GET /users/{userID}/partitions Gets a list of a users partitions

Parameters: {userID} = UserID GUID
 Returns: OK

Example Method:
 GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/partitions |
 {
 "Partitions": [
 {
 "PartitionID": "PartitionID GUID",
 "Description": "Description of the Partition"
 }
]
 }

POST /users{userID}/partition/{partitionID} Adds a partition to a user

Parameters: {userID} = UserID GUID
 {partitionID} = PartitionID GUID
 Returns: OK

Example Method:
 POST |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/partition/{partitionID} |

DELETE /users/{userID}/partition/{partitionID} Deletes a partition from a user

Parameters: {userID} = UserID GUID
 {partitionID} = PartitionID GUID
 Returns: OK

Example Method:
 DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/partition/{partitionID}

GET /users/{userID}/workstations Gets a list of a users workstations

Parameters: {userID} = UserID GUID
 Returns: OK
 ODATA Parameter: Query users with:
 ?\$filter=Name eq '{Workstation Name Text}'
 Return only Description fields with:
 ?\$select=Description

Example Method:

GET |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/workstations |
 {
 "Workstations": [
 {
 "WorkstationID": "WorkstationID GUID",
 "Name": "Name of Workstation",
 "Description": "Description of the Workstation",
 "Location": "Location of the Workstation"
 }
]
 }

POST /users/{userID}/workstations/{workstationID} Adds a workstation to a user

Parameters: {userID} = UserID GUID
 {workstationID} = WorkstationID GUID
 Returns: OK

Example Method:

POST |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/workstation/{workstationID} |

DELETE /users/{userID}/workstations/{workstationID} Deletes a workstation from a user

Parameters: {userID} = UserID GUID
 {workstationID} = WorkstationID GUID
 Returns: OK

Example Method:

DELETE |
 Endpoint: http://{Server}:{RestPort} |
 Resource: /pwapi/users/{userID}/workstation/{workstationID} |

GET /workstations Gets a list of workstations

Returns: OK (returns the list of users)
 ODATA Parameter: Query users with:
 ?\$filter=Name eq '{Name Text}'
 Return only Description field with:
 ?\$select=Description

Example Method:

GET |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/workstations` |

```
[
  {
    "WorkstationID": "WorkstationID GUID",
    "Name": "Name of the Workstation",
    "Description": "Description of the Workstation",
    "Location": "Location of the Workstation"
  }
]
```

POST **/workstations** **Adds a workstation**

Returns: OK

Returns the WorkstationID {ItemID} of the newly added record

Example Method:

POST |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/workstations` |

```
[
  {
    "Name": "Name of the Workstation",
    "Description": "Description of the Workstation",
    "Location": "Location of the Workstation"
  }
]
```

PUT **/workstations** **Updates a workstation**

Returns: OK

Example Method:

PUT |

Endpoint: `http://{Server}:{RestPort}` |

Resource: `/pwapi/workstations` |

```
[
  {
    "WorkstationID": "WorkstationID GUID",
    "Name": "Name of the Workstation",
    "Description": "Description of the Workstation",
    "Location": "Location of the Workstation"
  }
]
```

DELETE **/workstations/{wrkstID}** **Deletes a workstation**

Parameters: {workstationID} = WorkstationID GUID

Returns: OK

Example Method:

DELETE |

Endpoint: `http://{Server}:{RestPort}` |

Resource: /pwapi/workstations |

Get **/workstations/{wrkstID}/partitions****Gets a list of partitions for a workstation**

Parameters: {wrkstID} = WorkstationID GUID

Returns: OK

Example Method:

GET |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/workstations/{wrkstID}/partitions |

```
{
  "Partitions": [
    {
      "PartitionID": "PartitionID GUID",
      "Description": "Description of the Partition"
    }
  ]
}
```

POST **/workstations/{wrkstID}/partition/{partitionID}****Adds a partition to a**

Parameters: {wrkstID} = WorkstationID GUID

{partitionID} = PartitionID GUID

Returns: OK

Example Method:

POST |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/workstations/{wrkstID}/partition/{partitionID} |

DELETE **/workstations/{wrkstID}/partition/{partitionID}****Deletes a partition from a**

Parameters: {wrkstID} = WorkstationID GUID

{partitionID} = PartitionID GUID

Returns: OK

Example Method:

DELETE |

Endpoint: http://{Server}:{RestPort} |

Resource: /pwapi/workstations/{wrkstID}/partition/{partitionID} |

GET **/version****Get Pro-Watch, build, and API Version. OData parameters are enabled**

Returns: OK

```
{
  "ProWatchVersion": "Release 4.4",
  "ProWatchBuild": "11653",
  "DtuServiceVersion": "1.12.0.653"
}
```


SOAP Method Descriptions

Name	Description
AddAsset	Add a new asset
AddBadge	Add a new badge holder to the Pro-Watch database. Cards, clearance codes, individual doors and partitions can be added.
AddBadgeAsset	Add an asset to a badge holder
AddBadgeCert	Add a certification to a badge holder
AddBadgePartition	Assign a partition to a badge holder
AddCard	Add a card to a badge holder
AddCardByKeyField	Add a card to a badge from a key field (i.e. employee ID)
AddProvCardByKeyField	Add a provional card to a badge from a key field (i.e. employee ID)
AddCardClearCode	Add a clearance code to a card
AddProvCardClearCode	Add a clearance code to a provisional card
AddCardLogDevEx	Add a door exception to a card
AddTimezone	Add a timezone
AddUser	Add a user
AddUsersPartition	Add a partition to a user
AddUsersWorkStation	Add a workstation to a user
AddWorkStation	Add a workstation
AddWorkStationPartition	Add a partition to a workstation
CanCardAccessLogDev	Checks if card has access to the door right now.
CopyCard	Copy access from one card to another card for a badge holder.
CopyCardToProv	Copy access from one card to a provisional card for a badge holder.
DeleteAsset	Delete an asset
DeleteBadge	Delete a badge and all cards
DeleteBadgeAsset	Delete an asset from a badge holder
DeleteBadgeCert	Delete a certification from a badge holder
DeleteBadgePartition	Delete a partition from a badge holder
DeleteBlob	Delete a blob from an existing badge holder.
DeleteCard	Delete a card from an existing badge holder.
DeleteProvCard	Delete a provisional card from an existing badge holder.
DeleteCardClearCode	Delete a clearance code from a card.
DeleteCardClearCodes	Delete all clearance codes for a card.
DeleteCardLogDevEx	Delete a door exception for a card
DeleteUser	Delete a user
DeleteUsersPartition	Delete a partition from a user
DeleteUsersWorkStation	Delete a workstation from a user
DeleteWorkStation	Delete a workstation
DeleteWorkStationPartition	Delete a partition from a workstation
GetAreaOccupants	Gets occupants currently in Pro-Watch area
GetAreas	Returns a list of all Pro-Watch areas
GetAssets	Returns a list of all assets
GetAuditLogEntries	Returns a list of audit log entries
GetAuditLogPublishedEntries	Returns a list of audit log entries that are from tables marked as published
GetAuditLogPublishedTables	Returns a list of tables that are marked as being published.
GetBadge	Using a Pro-Watch unique identifier, this will return all badge fields for a badge holder. All data from BADGE, BADGE_V.
GetBadgeAssets	Get all assets for a badge holder
GetBadgeBlob	Get a badge holder blob/photo/signature using a Pro-Watch unique identifier

GetBadgeBlobByKeyField	Get the badge holders blob/picture or whatever the blob is. A badge key field is used to get the badge holder.
GetBadgeByCard	Using a Pro-Watch card number, this will get all badge fields for a badge holder. All data from BADGE, BADGE_V.
GetBadgeByKeyField	This method will return all badge fields from BADGE and BADGE_V using a badge field value. For example, if there is a unique field in BADGE_V called EMPLOYEE_ID, then this method could be called using a badge holder employee ID.
GetBadgeCerts	Returns a list of certifications
GetBadgePhoto	Returns the badge photo for a badge holder Note: This method uses a default badge photo property in the DTU Service configuration file to get the correct blob
GetBadgePhotoByKeyField	Returns the badge holder photo
GetBadges	Returns a list of all badge holders in the Pro-Watch database
GetBadgesBlobs	Return a paged list of all badge holder photos
GetBadgesCards	Return a paged list of all badges and their cards, clearance codes, door exceptions and partitions.
GetBadgesModified	Returns a list of badges with card data modified between two dates
GetBadgesPaging	Returns a list of badges based on page size and page number
GetBadgeSchema	Returns a list of all Pro-Watch badge fields
GetBadgeSchemaDD	Returns a list of drop down values for a drop down badge field
GetBadgeStatuses	Returns a list of badge statuses
GetBadgeTypes	Returns a list of all badge types
GetBlobTypes	Returns a list of all blob types
GetCard	Get card information for a badge holder using a card number
GetCards	Returns a list of cards for a badge holder
GetProvCards	Returns a list of provisional cards for a badge holder
GetCertifications	Returns a list of all certifications
GetChannels	Get a list of all channels
GetChannelsBySite	Get a list of all channels for a Pro-Watch site
GetClearanceCodes	Get a list of all clearance codes
GetClearanceCodesLD	Get a list of all logical devices for a clearance code
GetCompanies	Get all companies
GetCompany	Get company detail information
GetCompanyContacts	Get company contacts for a company
GetLogicalDeviceHardware	Lists all the actual hardware under a given Pro-Watch Logical device.
GetLogicalDevices	Lists all the Pro-Watch Logical Devices under a given Site. (For smaller sites it could get all associated hardware under a given Pro-Watch Logical device as well)
GetLogicalDevicesAll	Lists all Pro-Watch logical devices in the entire system.
GetLogDevsByHWClass	Get all logical devices for a hardware class
GetPanels	Get a list of all panels
GetPanelsBySite	Get all panels under a give site.
GetPanelTimeZones	Get a list of time zones for a panel
GetPartitions	Get a list of all partitions
GetPrograms	Returns a list of programs and functions that can and cannot be accessed
GetProvCards	Returns a list of all provisional cards
GetSites	Gets a list of all Pro-Watch hardware sites.
GetSubpanels	Gets list of all subpanels.
GetSubpanelsBySite	Gets list of all subpanels under a give site.

GetTimezones	Gets a list of all time zones and schedules for the timezone.
GetUsers	Get a list of users
GetUsersPartitions	Get a list of partitions for a user
GetUsersWorkStations	Get a list of workstations for a user
GetWorkStations	Get a list of workstations
GetWorkStationPartitions	Get a list of partitions for a workstation
GetVersion	Returns the Pro-Watch version and build. Also returns the version of the DTU Service.
HardwareLock	Lock a door/logical device
HardwareMomentaryUnlock	Momentary unlock a door/logical device
HardwareReEnable	Re-enable a logical device
HardwareTimedOverride	Timed override a door/logical device
HardwareUnlock	Unlock a door/logical device
IssueAnEvent	Issues an event as specified by event parameters
IssueChannelEvent	Issue a channel event from the addressed channel.
IssueCtrlEvent	Issue a CTRL event in Pro-Watch
IssueInputEvent	Issue an Input event from the addressed Input.
IssueOutputEvent	Issue an Output event from the addressed Output.
IssuePanelEvent	Issue a Panel event from the addressed Panel.
IssueSubpanelEvent	Issue a Subpanel event from the addressed Subpanel.
QueryBadges	Gets a list of badges from Pro-Watch. A list of filters can be sent in to limit the data returned.
QueryBadgesPaging	Same as QueryBadges, but with paging for page size and page
QueryCards	Get a list of cards from Pro-Watch. A list of filters can be sent in to limit the data returned.
QueryCardsPaging	Same as QueryCards, but with paging for page size and page
QueryCardsWithBadgePaging	Same as QueryCards, but with paging for page size and page, and also returns badge information with the cards
SetBadgeBlob	Set a badge holder phot/signature/electronic document using the Pro-Watch unique identifier
SetBadgePhoto	Add or update a badge photo Note: This method uses a default badge photo property in the DTU Service configuration file to get the correct blob
SetBadgePhotoByKeyField	Adds or updates a photo for a badge holder
TerminateBadgeCard	Terminate a badge and active cards. This will set the badge status to TERMINATED, set badge expire date to current date, terminate all active cards, set the card expiration date to the current date for all active cards and remove the clearance codes from all cards.
UnlockIfCardCanAccessLogDev	Checks if the card has access to the door right now and momentary unlocks the door.
UpdateAsset	Update the description of an asset
UpdateBadgeByKeyField	Update a badge holder. Cards, clearance codes and individual doors can be modified.
UpdateBadge	Update a badge holder using a the Pro-Watch unique badge ID
UpdateBadgeAsset	Update an asset issued to a badge holder
UpdateBadgeCert	Update a badge holder certification
UpdateCard	Update an existing card.
UpdateCardLogDevEx	Update a door exception
UpdateTimezone	Update an existing timezone

SOAP Method Return

All SOAP methods return:

TransStatus	SUCCESS or FAIL	If the method call is successful, this value will be SUCCESS. If the method call fails in any way, this value will be FAIL.
TransStatusText	string	If the TransStatus = FAIL, this will give the reason for the failure.
TransNumber	short	Some calls may return a TransNumber value.

Method Details

Method: **AddAsset** (PwAsset asset)

This method will add a new asset to the Pro-Watch database. An asset is simply the name of something, a key, a cell phone, etc that will get issued to a badge holder.

Input Parameters

PwAsset	A new Pro-Watch asset. Currently it is just the name of the asset
---------	---

Return Value

PwStatusAddAsset	This will return the Pro-Watch unique identifier of the new asset
------------------	---

Method: **AddBadge** (PwBadge badge)

Add a new badge holder to the Pro-Watch database. Cards, clearance codes and individual doors can be added. If any active cards are added, it will be automatically downloaded to the controllers.

When adding a new Pro-Watch badge holder, only the Last Name is required. If badge issue date is not supplied, it will be defaulted to the current date/time. If a badge expire date is not supplied, it will be defaulted to 01/01/2100.

When adding cards, a unique card number and Pro-Watch Company is required. If no card status is supplied, the card will default to Active. If card issue date is not supplied, it will be defaulted to the current date/time. If a card expire date is not supplied, it will be defaulted to one year from the current date/time.

Input Parameters

PwBadge	A badge structure that allows for all badge, card, clearance codes and door exception data to be added to Pro-Watch
---------	---

Return Value

PwStatusAddBadge	This will return the Pro-Watch unique identifier of the new badge holder as well as any Pro-Watch custom auto-increment badge field values. If cards are added, this will also return all card creation statuses (PwStatusCard).
------------------	--

Method: AddBadgeAsset (PwBadgeAsset asset)

This method will add an asset to a badge holder. Required fields are badge ID and asset ID. Key number is optional and should be used for a key that has many copies but the copies may be numbered or have some type of serial number to differentiate the copies.

Note: Asset statuses are defined in a table valued function called PWAP_KEY_STATUSES. Always send in the text not the code.

Input Parameters

PwBadgeAsset	The asset, status and various dates to add
--------------	--

Return Value

PwStatus	Pass or Fail
----------	--------------

Method: AddBadgeCert (PwBadgeCert certification)

This method will add a certification to a badge holder. Certification date is required to calculate how long the certification is valid for.

Note: List of attempts comes from database function PWAP_CERTIFICATION_ATTEMPTS. List of results from database function PWAP_CERTIFICATION_RESULTS.

Input Parameters

PwBadgeCert	The certification to add to the badge holder
-------------	--

Return Value

PwStatus	Pass or Fail
----------	--------------

Method: AddBadgePartition (string badgeID, PwPartition partition)

This method will assign a badge holder to a partition.

A partition is a way to manage what data certain Pro-Watch users are allowed to see. For instance, if you have multiple sites throughout the country, you can partition each site so site users only see their own site data, where as an administrator can see all sites.

Input Parameters

PwPartition	The partition to assign to the badge holder
-------------	---

Return Value

PwStatus	Pass or Fail
----------	--------------

Method: AddCard (PwCard newCard)

This method will add a card to a badge holder. Use this method when you know the unique Pro-Watch badge ID of the person.

Fields required to add a card:

Badge ID	This is the Pro-Watch unique identifier for the badge holder that the card will be added. Ex. 0x00294637433374146412D333245362D3439
Card Number	This must be a unique card number in the Pro-Watch database
Company	This must be an existing company in the Pro-Watch database. This value can be the name of the company or the Pro-Watch unique ID for the company.

Other card field notes:

Status Code	If not supplied, the card will be default to 'Active'
Issue Date	If not supplied, the default card issue date will be set to the current date and time.
Expire Code	If not supplied, the default card expire date will be one year from the current date.
Card Type	This is the physical look of the card from Badge Types

Input Parameters

PwCard	A card structure that can include clearance codes and door exceptions
--------	---

Return Value

PwStatusCard	Returns the status of the card. Card number if sequential, card downloaded.
--------------	---

Method: AddCardByKeyField (PwCard newCard, PwCustomBadgeField keyField)

This method will add a card to a badge holder using a custom badge field. The key field, such as an Employee ID, is used to look up the badge holder so it knows who to add the card to.

Fields required to add a card:

Card Number	This must be a unique card number in the Pro-Watch database
Company	This must be an existing company in the Pro-Watch database. This value can be the name of the company or the Pro-Watch unique ID for the company.

Other card field notes:

Status Code	If not supplied, the card will be set to 'Active'
Issue Date	If not supplied, the default card issue date will be set to the current date and time.
Expire Code	If not supplied, the default card expire date will be one year from the current date.

Input Parameters

PwCard	A card structure that can include clearance codes and door exceptions
PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.

Return Value

PwStatusCard	Returns the status of the card. Card number if sequential, card downloaded.
--------------	---

Method: **AddProvCardByKeyField** (PwProvCard newCard, PwCustomBadgeField keyField)

This method will add a card to a badge holder using a custom badge field. The key field, such as an Employee ID, is used to look up the badge holder so it knows who to add the card to.

Fields required to add a card:

Company	This must be an existing company in the Pro-Watch database. This value can be the name of the company or the Pro-Watch unique ID for the company.
---------	---

Other card field notes:

Status Code	If not supplied, the card will be set to 'Active'
Issue Date	If not supplied, the default card issue date will be set to the current date and time.
Expire Code	If not supplied, the default card expire date will be one year from the current date.
Provision Desc	Description of the provisional card

Input Parameters

PwCard	A card structure that can include clearance codes and door exceptions
PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.

Return Value

PwStatusCard	Returns the status of the card. Provisional Card number.
--------------	--

Method: **AddCardClearCode** (string cardNumber, PwClearCode[] newClearCodes)

This method will add an array of clearance codes to a card.

Input Parameters

string	The card number the clearance code will be added to.
PwClearCode	An array of clearance codes to be added.

Return Value

PwStatusCardCC	Status of the operation
----------------	-------------------------

Method: **AddProvCardClearCode** (string cardNumber, PwClearCode[] newClearCodes)

This method will add an array of clearance codes to a card.

Input Parameters

string	The provional card number the clearance code will be added to.
PwClearCode	An array of clearance codes to be added.

Return Value

PwStatusCardCC	Status of the operation
----------------	-------------------------

Method: **AddCardLogDevEx** (PwCardLogDevEx logDevEx)

This method will add a door exception to a card. Door exceptions can be granted or removed. A Granted door means they have access to this door outside any normal clearance code access. A removed door means they will not get through that door even if they have it granted through a clearance code. If a door exception is granted, a valid time zone is required. For a list of valid time zones for a door, use the GetPanelTimeZones method.

Note: A door exception can be temporarily granted. Set the TempAccess flag and set the start and end date/time.

Parameters

PwCardLogDevEx	Parameters used to add the granted or revoked door exception
----------------	--

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **AddTimezone** (PwTimeZone pwTimeZone)

This method adds a new timezone (with schedules) in Pro-Watch

Parameters

PwTimeZone	Timezone (with schedules) to be added
------------	---------------------------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **AddUser** (PwUser pwUser)

This method adds a new user in Pro-Watch

Parameters

PwUser	User to be added
--------	------------------

Return Value

PwStatusAddItem	Success or Fail, ItemID representing the userID (GUID) of the user added
-----------------	--

Method: **AddUserPartition** (string userID, string partitionID)

This method adds a partition to a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to add the partition to
partitionID	partitionID (GUID) of the partition to add to the user

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **AddUserWorkstation** (string userID, string workstationID)

This method adds a workstation to a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to add the workstation to
workstationID	workstationID (GUID) of the workstation to add to the user

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **AddWorkstation** (PwWorkstation pwWorkstation)

This method adds a new workstation in Pro-Watch

Parameters

PwWorkstation	workstation to be added
---------------	-------------------------

Return Value

PwStatusAddItem	Success or Fail, ItemID representing the workstationID (GUID) of the workstation added
-----------------	--

Method: **AddWorkstationPartition** (string workstationID, string partitionID)

This method adds a partition to a workstation in Pro-Watch

Parameters

workstationID	workstationID (GUID) of the workstation to add the partition to
partitionID	partitionID (GUID) of the partition to add to the workstation

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **CanCardAccessLogDev** (String LogDevID, String CardNo, String CurrDateTime)

This method will check if an existing card has access to the given door right now.

Parameters

LogDevID	LogicalDevID GUID
CardNo	CardNo from PW Database
CurrDateTime	Date and time with format YYYY-MM-DDThh:mm:ss

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **CopyCard** (PwCopyCardParams copyCardParams)

This method will copy an existing card to a new card for a badge holder. The new issue and expire date can be set as well as disabling the card that was copied.

Parameters

PwCopyCardParameters	Parameters that drive the copying of the card. (copy card number, new card number, issue date, expire date, disable old/copy card)
----------------------	--

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **CopyCardToProv** (PwCopyCardToProvParams copyCardParams)

This method will copy an existing card to a new card for a badge holder. The new issue and expire date can be set as well as disabling the card that was copied.

Parameters

PwCopyCardToProvParameters	Parameters that drive the copying of the card. (copy card number, issue date, expire date, disable old/copy card, provision desc)
----------------------------	---

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **DeleteAsset** (string assetID)

This method will delete a Pro-Watch asset. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The Pro-Watch asset ID to be deleted.
--------	---------------------------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteBadge (string badgeID)

This method will delete a badge and cards for that badge holder from the Pro-Watch database. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The Pro-Watch badge ID of the badge to be deleted.
--------	--

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteBadgeAsset (string badgeID, string assetID, string assetNo)

This method will delete an asset from a badge holder. If asset numbers are not used, set asset number to blank. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The badge ID of the badge holder
string	The asset ID to be removed from badge holder
string	The unique number of the asset to be removed from badge holder

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteBadgeCert (string badgeID, string rowID)

This method will delete a certification from a badge holder. Because this table maintains history, the rowID is the unique row value for the certification to be deleted.

Input Parameters

string	The badge ID of the badge holder
string	The unique row ID of the badge certification table

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteBadgePartition (string badgeID, string partitionID)

This method will delete a partition from a badge holder.

Input Parameters

string	The badge ID of the badge holder
string	The partition ID of the partition to delete

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteBlob (string badgeID, string blobTypeID)

This method will delete a blob from a badge holder. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The badge ID of the badge holder
String	The blob type ID of the blob to delete

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteCard (string cardNumber)

This method will delete a card from a badge holder. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The card number to be deleted.
--------	--------------------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: DeleteProvCard (string cardNumber)

This method will delete a provisional card from a badge holder. This operation will be logged in the Pro-Watch Audit Log.

Input Parameters

string	The provional card number to be deleted.
--------	--

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: `DeleteCardClearCode` (`string` cardNumber, `PwClearCode[]` deleteClearCodes)

This method will delete an array of clearance codes from a card. This operation will be audited in the Pro-Watch Audit Log.

Input Parameters

string	The card number the clearance code will be deleted from.
PwClearCode	An array of clearance codes to be deleted.

Return Value

PwStatus	Status of the operation
----------	-------------------------

Method: `DeleteCardClearCodes` (`string` cardNumber)

This method will delete all clearance codes for a card. This operation will be audited in the Pro-Watch Audit Log.

Input Parameters

string	The card number the clearance code will be deleted from.
--------	--

Return Value

PwStatus	Status of the operation
----------	-------------------------

Method: `DeleteCardLogDevEx` (`string` cardNumber, `PwLogDev` logDev)

This method will delete a door exception from a card. This operation will be audited in the Pro-Watch Audit Log.

Input Parameters

string	The card number the door exception will be deleted from.
PwLogDev	The door/logical device to be removed from the card

Return Value

PwStatus	Status of the operation
----------	-------------------------

Method: `DeleteUser` (`string` userID)

This method Deletes a user in Pro-Watch

Parameters

PwUser	User to be Deleted
--------	--------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **DeleteUserPartition** (string userID, string partitionID)

This method deletes a partition from a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to delete the partition
partitionID	partitionID (GUID) of the partition to delete from the user

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **DeleteUserWorkstation** (string userID, string workstationID)

This method deletes a workstation from a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to delete the workstation from
workstationID	workstationID (GUID) of the workstation to delete from the user

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **DeleteWorkstation** (string workstationID)

This method deletes a workstation in Pro-Watch

Parameters

PwWorkstation	workstation to be deleted
---------------	---------------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **DeleteWorkstationPartition** (string workstationID, string partitionID)

This method deletes a partition from a workstation in Pro-Watch

Parameters

workstationID	workstationID (GUID) of the workstation to delete the partition from
partitionID	partitionID (GUID) of the partition to delete from the workstation

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: **GetAreaOccupants** (string areaID)

Returns a list of Pro-Watch card holders who are currently in a area.

Input Parameters

string	The unique ID/GUID of the Pro-Watch area
--------	--

Return Value

PwStatusGetAreaOccupants	Returns a list of people/cards currently in a Pro-Watch area
--------------------------	--

Method: **GetAssets**

Returns a list of Pro-Watch assets.

Return Value

PwStatusGetAssets	Returns a list of Pro-Watch assets
-------------------	------------------------------------

Method: **GetAuditLogEntries** (int pageSize, int page, DateTime startDate, DateTime endDate)

Returns a list of Pro-Watch audit log entries.

Input Parameters

pageSize	The number of entries to return
Page	The page of entries to return
startDate	The start of the date range to return entries from
endDate	The end of the data range to return entries from

Return Value

PwStatusGetLogEntries	Returns a list of Pro-Watch AuditLog Entries
-----------------------	--

Method: **GetAuditLogPublishedEntries** (int pageSize, int page, DateTime startDate, DateTime endDate)

Returns a list of Pro-Watch audit log entries that are from tables marked as being published.

Input Parameters

pageSize	The number of entries to return
Page	The page of entries to return
startDate	The start of the date range to return entries from

endDate	The end of the data range to return entries from
---------	--

Return Value

PwStatusGetLogEntries	Returns a list of Pro-Watch AuditLog Entries
-----------------------	--

Method: **GetAuditLogPublishedTables** ()

Returns a list of Pro-Watch tables that are marked as being published.

Return Value

PwStatusGetPublishedTables	Returns a list of Pro-Watch Tables that are marked as being published.
----------------------------	--

Method: **GetAreas** (bool getLogDevData)

Returns a list of Pro-Watch areas as well as logical devices associated with the area.

Input Parameters

boolean	If true, this method will return all areas as well as the In/Out logical devices associated with the area. If false, only areas will be returned
---------	--

Return Value

PwStatusGetAreas	Returns a list of all Pro-Watch areas
------------------	---------------------------------------

Method: **GetBadge** (string badgeID, bool getCardData)

Using a Pro-Watch unique identifier, this will get all badge fields for a badge holder. All data from BADGE, BADGE_V.

Input Parameters

string	The Pro-Watch badge holder unique identifier. Must be in the form of 0x002942324142413433312D303244432D3432.
boolean	If true, this method will return all card data including clearance codes and door exceptions. If false, only badge data will be returned.

Return Value

PwStatusGetBadge	Returns a list of badges for badge holder
------------------	---

Method: **GetBadgeAssets** (string badgeID)

This will get all assets for a badge holder.

Input Parameters

string	The Pro-Watch badge holder unique identifier. Must be in the form of 0x002942324142413433312D303244432D3432.
--------	--

Return Value

PwStatusGetBadgeAssets	Returns a list of assets for the badge holder
------------------------	---

Method: **GetBadges**

Returns a list of all badge holders in the Pro-Watch database. All data from BADGE, BADGE_V.

Note: SoapUI crashed when trying to return a list of 500 badges. Fiddler returned all 500 without a problem.

Return Value

PwStatusGetBadge	Returns a list of all Pro-Watch badges
------------------	--

Method: **GetBadgeBlob**(string keyField, PwBlobType blobType)

This method will get a badge holder blob/picture or whatever the blob is using the Pro-Watch unique identifier. The blobType can be either the blob type description or the Pro-Watch blob ID.

Input Parameters

string	Pro-Watch badge holder unique identifier. Must be in the form of 0x002942324142413433312D303244432D3432
PwBlobType	This is a blob type structure that identifies the description of the Pro-Watch blob type or the Pro-Watch unique identifier of the blob type.

Return Value

PwStatusGetBlob	The blob/photo/electronic document if found
-----------------	---

Method: **GetBadgesBlobs**(int pageSize, int page)

Returns a list of badge holders based on a page size and page number. For each badge returned, the primary photo of the badge holder will be returned.

This method uses a default Pro-Watch blob types defined in the .config file of this application.

Input Parameters

int	The number of badges to be returned
int	The page number of badges to return

Return Value

PwStatusGetBadgeBlobs	Returns a list of all Pro-Watch badge holder photos
-----------------------	---

Method: **GetBadgesCards**(int pageSize, int page)

Returns a list of badge holders based on a page size and page number. For each badge returned, the cards, clearance codes, door exceptions and partitions will be returned for the badge.

Input Parameters

int	The number of badges to be returned
int	The page number of badges to return

Return Value

PwStatusGetBadge	Returns a list of all Pro-Watch badges
------------------	--

Method: **GetBadgesModified**(DateTime startDate, DateTime endDate)

Returns a list of badge holders (with card data) who have been modified between the start and end date. This method will return a badge when badge, card, clearance codes, door exceptions, or a photo has been modified.

Input Parameters

DateTime	The start date
DateTime	The end date

Return Value

PwStatusGetBadge	Returns a list of Pro-Watch badges modified between the start and end date
------------------	--

Method: **GetBadgesPaging**(int pageSize, int page)

Returns a list of badge holders based on a page size and page number. For instance, if we have 1000 badge holders, GetBadgePaging(100, 2) will return a list of 100 badges from row 101 to 200, sorted by last name, first name, middle name. See QueryBadgesPaging for adding filters and other sorting.

Input Parameters

int	The number of badges to be returned
int	The page number of badges to return

Return Value

PwStatusGetBadge	Returns a list of all Pro-Watch badges
------------------	--

Method: **GetBadgeBlobByKeyField**(PwCustomBadgeField keyField, PwBlobType blobType)

This method will get a badge holder blob/picture or whatever the blob is. A badge key field is used to get the badge holder BLOB. The blobType can be either the blob type description or the Pro-Watch blob ID.

Input Parameters

PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.
PwBlobType	This is a blob type structure that identifies the description of the Pro-Watch blob type or the Pro-Watch unique identifier of the blob type.

Return Value

PwStatusGetBlob	The blob/photo if found
-----------------	-------------------------

Method: **GetBadgeByCard**(string cardNumber, bool getCardData)

Using a Pro-Watch card number, this will get all badge fields for a badge holder. All data from BADGE, BADGE_V.

Input Parameters

cardNumber	A valid Pro-Watch card number
getCardData	If true, this method will return all card data including clearance codes and door exceptions. If false, only badge data will be returned.

Return Value

PwStatusGetBadge	Returns a list of badges found
------------------	--------------------------------

Method: **GetBadgeByKeyField**(PwCustomBadgeField keyField, bool getCardData)

This method will return all badge fields from BADGE and BADGE_V using a badge field value. For example, if there is a unique field in BADGE_V called EMPLOYEE_ID, then this method could be called using a badge holders employee ID.

Input Parameters

PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.
boolean	If true, this method will return all card data including clearance codes and door exceptions. If false, only badge data will be returned.

Return Value

PwStatusGetBadge	Returns a list of badges found
------------------	--------------------------------

Method: **GetBadgeCerts** (string badgeID)

This method will return all certifications for a badge holder.

Input Parameters

string	The Pro-Watch badge holder unique identifier. Must be in the form of 0x002942324142413433312D303244432D3432.
--------	--

Return Value

PwStatusGetBadgeCerts	Returns a list of certifications for the badge holder
-----------------------	---

Method: **GetBadgePhoto** (string badgeID)

Get a badge holder blob/picture using the unique Pro-Watch badge ID.

This method uses a default Pro-Watch blob types defined in the .config file of this application.

Input Parameters

badgeID	The Pro-Watch badge unique identifier in the form 0x002946374333374146412D333245362D3439
---------	--

Return Value

PwStatusGetBlob	The blob/photo in base64 format if found.
-----------------	---

Method: **GetBadgePhotoByKeyField** (PwCustomBadgeField keyField)

Get a badge holder blob/picture. A badge key field is used to get the badge holder BLOB.

This method uses a default Pro-Watch blob type defined in the .config file of this application.

Input Parameters

PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.
--------------------	---

Return Value

PwStatusGetBlob	The blob/photo in base64 format if found.
-----------------	---

Method: **GetBadgeSchema** ()

This method returns the badge schema for a Pro-Watch database including badge column name, display name, length and other such properties.

Return Value

PwStatusGetBadgeSchema	A list of all the badge fields for the Pro-Watch database. Properties include, column name, display name, length, resource type, etc
------------------------	--

Method: **GetBadgeSchemaDD** (string columnName)

This method returns a list of all values for a Pro-Watch drop down badge field. The database column name of the badge field is used to get the drop down values. Call GetBadgeSchema first and then for each drop down field returned, use this method to get the list of drop down values.

Input Parameters

string	The name of the drop down badge field column name
--------	---

Return Value

PwStatusGetBadgeSchemaDD	A list of all the drop down values
--------------------------	------------------------------------

Method: GetBadgeStatuses

This method will return a list of all the badge statuses in Pro-Watch.

Return Value

PwStatusGetBadgeStatuses	A list of all Pro-Watch Badge Statuses
--------------------------	--

Method: GetBadgeTypes

This method will return a list of all the badge types in Pro-Watch.

Return Value

PwStatusGetBadgeTypes	A list of all Pro-Watch Badge Types
-----------------------	-------------------------------------

Method: GetBlobgTypes

This method will return a list of all the blob types in Pro-Watch.

Return Value

PwStatusGetBlobTypes	A list of all Pro-Watch Blob Types
----------------------	------------------------------------

Method: GetCard (string cardNumber)

This method will use a card number to return all information for a card. All the clearance codes and door exceptions for the card will be returned.

Input Parameters

string	The card number of the card object to get
--------	---

Return Value

PwStatusGetCard	The card object with all the card information
-----------------	---

Method: GetCards (string badgeID)

This method will use a badge ID to return all cards for a badge holder. All clearance codes and door exceptions for each card will be returned.

Input Parameters

string	The Pro-Watch badge unique identifier in the form 0x002946374333374146412D333245362D3439
--------	---

Return Value

PwStatusGetCards	A list of all cards for a badge holder
------------------	--

Method: GetProvCards (string badgeID)

This method will use a badge ID to return all provional cards for a badge holder. All clearance codes and door exceptions for each card will be returned.

Input Parameters

string	The Pro-Watch badge unique identifier in the form 0x002946374333374146412D333245362D3439
--------	--

Return Value

PwStatusGetCards	A list of all cards for a badge holder
------------------	--

Method: GetCertifications

This method will return a list of all Pro-Watch certifications.

Return Value

PwStatusGetCertifications	A list of all Pro-Watch certifications
---------------------------	--

Method: GetChannels

This method will return a list of all Pro-Watch channels.

Return Value

PwStatusGetChannels	A list of all Pro-Watch channels
---------------------	----------------------------------

Method: GetChannelsBySite (string siteID)

This method will return a list of channels for a given Pro-Watch site ID.

Input Parameters

string	The Pro-Watch site ID of the channels to get
--------	--

Return Value

PwStatusGetChannels	A list of all Pro-Watch channels for the input site ID
---------------------	--

Method: GetClearanceCodes

This method will return a list of all Pro-Watch clearance codes/access groups.

Return Value

PwStatusGetClearCodes	A list of all Pro-Watch clearance codes/access groups
-----------------------	---

Method: `GetClearanceCodesLD` (`string` clearCodeID)

This method returns a list of logical devices assigned to the given clearance code.

Input Parameters

string	The clearance code ID of the logical devices to get
--------	---

Return Value

PwStatusGetClearCodesLD	Returns a list of logical devices assigned to the clearance code
-------------------------	--

Method: `GetCompany` (`string` companyID)

This method returns company detail for a Pro-Watch company.

Input Parameters

string	The Pro-Watch company unique identifier in the form of 0x00481413AFD6158242BB802BAA719B621B77
--------	---

Return Value

PwStatusGetCompanies	The first item in a company list
----------------------	----------------------------------

Method: `GetCompanies`

This method will return a list of all Pro-Watch companies.

Return Value

PwStatusGetCompanies	A list of all Pro-Watch Companies
----------------------	-----------------------------------

Method: `GetCompanyContacts` (`string` companyID)

This method returns a list of company contacts for a Pro-Watch company.

Input Parameters

string	The Pro-Watch company unique identifier in the form of 0x00481413AFD6158242BB802BAA719B621B77
--------	---

Return Value

PwStatusGetCoContacts	A list of company contacts for the company
-----------------------	--

Method: `GetLogicalDevices` (`string` SiteID, `bool` IncludeLogDevHw)

This method gets all the Pro-Watch Logical Devices under a given site. The Site identifier is obtained through a call to `GetAllSites()`. If the flag `IncludeLogDevHw` is set to true then it gets details of all of the actual hardware under the Logical device grouping as well.

Input Parameters

string	This is a Unique Site Identifier.
boolean	If this flag is true it gets details of all hardware under the logical device as well.

Return Value

PwStatusGetLogDevices	Contains list of Logical devices under site.
-----------------------	--

Method: GetLogicalDevicesAll

This method gets all the Pro-Watch Logical Devices in the System.

Return Value

PwStatusGetLogDevices	Contains list of Logical devices in the System.
-----------------------	---

Method: GetLogicalDeviceHardware (string LogDevID)

This method gets details of all Hardware Devices under a given Logical Device grouping.

Input Parameters

string	This is a Unique Logical device identifier.
--------	---

Return Value

PwStatusGetLogDeviceHardware	Contains list of actual hardware under a given logical device.
------------------------------	--

Method: GetLogDevByHWClass (string hwClass, bool logDevHW)

This method gets details of all Hardware Devices under a given Logical Device grouping.

Input Parameters

string	The hardware class id
bool	If this flag is true it gets details of all hardware under the logical device as well.

Return Value

PwStatusGetLogDevices	List of logical devices in the hardware class.
-----------------------	--

Method: GetPanels

This method will return a list of all Pro-Watch panels.

Return Value

PwStatusGetPanels	A list of all Pro-Watch panels
-------------------	--------------------------------

Method: **GetPanelsBySite** (string siteID)

This method will return a list of panels for a given Pro-Watch site ID.

Input Parameters

string	The Pro-Watch site ID of the channels to get
--------	--

Return Value

PwStatusGetPanels	A list of all Pro-Watch panels for the input site ID
-------------------	--

Method: **GetPanelTimeZones** (string panelID)

This method will return a list of time zones in a panel.

Input Parameters

string	The panel ID
--------	--------------

Return Value

PwStatusGetTimezones	A list of all time zones for the panel
----------------------	--

Method: **GetPartitions**

This method will return a list of all Pro-Watch partitions.

Return Value

PwStatusGetPartitions	A list of all Pro-Watch partitions
-----------------------	------------------------------------

Method: **GetPrograms**

This method will return a list of Pro-Watch programs and functions that can and cannot be accessed through the DTU Service.

Return Value

PwStatusGetPrograms	A list programs and functions
---------------------	-------------------------------

Method: **GetSites**

Pro-Watch System is set up to group physical hardware like Readers, Door position switches, Input and Output points among others in to Logical Devices. There could exist one or more Logical devices in a given location/region called Site.

This function call gets all the Sites in the Pro-Watch system. This method is used in conjunction with few others to drill down to a specific hardware device under a Site.

Return Value

PwStatusGetSites	A list of all sites.
------------------	----------------------

Method: **GetSubpanels**

This method will return a list of all Pro-Watch Subpanels.

Return Value

PwStatusGetSubpanels	A list of all Pro-Watch Subpanels
----------------------	-----------------------------------

Method: **GetSubpanelsBySite** (string siteID)

This method will return a list of subpanels for a given Pro-Watch site ID.

Input Parameters

string	The Pro-Watch site ID of the Subpanels to get
--------	---

Return Value

PwStatusGetSubpanels	A list of all Pro-Watch Subpanels for the input site ID
----------------------	---

Method: **GetTimezones**

Returns a list of all time zones and schedules for each time zone

Return Value

PwStatusGetTimezones	A list of all time zones
----------------------	--------------------------

Method: **GetUsers** ()

This method gets a list of users in Pro-Watch

Return Value

PwStatusGetUsers	Success or Fail
------------------	-----------------

Method: **GetUsersPartitions** (string userID)

This method gets a list of partitions for a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to get the partition
--------	--

Return Value

PwStatusGetPartitions	Success or Fail
-----------------------	-----------------

Method: `GetUsersWorkstations` (string userID)

This method gets a list of workstations for a user in Pro-Watch

Parameters

userID	userID (GUID) of the user to get the workstations for
--------	---

Return Value

PwStatusGetWorkstation	Success or Fail
------------------------	-----------------

Method: `GetWorkstations` ()

This method gets a list of workstations in Pro-Watch

Return Value

PwStatusGetWorkstation	Success or Fail
------------------------	-----------------

Method: `GetWorkstationPartitions` (string workstationID)

This method gets a list of partitions for a workstation in Pro-Watch

Parameters

workstationID	workstationID (GUID) of the workstation to get the partitions for
---------------	---

Return Value

PwStatusGetPartitions	Success or Fail
-----------------------	-----------------

Method: `GetVersion`

Returns the Pro-Watch version and build as well as the version of the DTU Service

Return Value

PwStatusGetVersion	Return Pro-Watch version, Pro-Watch build, DTU Service version
--------------------	--

Method: `HardwareLock` (string logicalDeviceID)

This method will lock a door/logical device. When locked, no one will be able to get through the door. This method requires the additional DTU API license option.

Input Parameters

string	The door/logical device ID
--------	----------------------------

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **HardwareMomentaryUnlock** (string logicalDeviceID)

This method will momentary unlock a door/logical device. A momentary unlock is when a door is opened for the same time as is opened when someone swipes a card. This method requires the additional DTU API license option.

Input Parameters

string	The door/logical device ID
--------	----------------------------

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **HardwareReenable** (string logicalDeviceID)

This method will re-enable a door/logical device. This method requires the additional DTU API license option.

Input Parameters

string	The door/logical device ID
--------	----------------------------

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **HardwareTimedOverride** (string logicalDeviceID, int minutes)

This method will time override a door/logical device. The door will be opened and alarms will be shunted for X number of minutes. This method requires the additional DTU API license option.

Input Parameters

string	The door/logical device ID
int	The number of minutes to mask door events

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **HardwareUnlock** (string logicalDeviceID)

This method will unlock a door/logical device. This method requires the additional DTU API license option.

Input Parameters

string	The door/logical device ID
--------	----------------------------

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: IssueAnEvent (PwIssueEvent EventMsg)

This method issues an event in the Pro-Watch system, that is seen by all clients (if routing group allows it), of the specified type and tied to the addressed hardware device. (**Note:** The event type specified in the parameter should exist in Pro-Watch and be a part of the events supported by the addressed hardware device. The hardware device need not be online but is required to be created for addressing purpose. The *Channel* needs to be *installed* at the time of creation and can be later *un-installed* after the hardware tree has been created.)

The “**Issue Event**” permission should be assigned to a Pro-Watch user to give the user the ability to perform this action. To add the “Issue Event” permission for a Pro-Watch user, go to the Program tab under User’s Properties, select **Programs->Administration->Data Transfer Utility** and click on “**Add Function...**” button and select “**Issue Event**” function to assign the permission to the user.

Input Parameters

PwIssueEvent	Depending on the event issued the necessary fields include Hardware ID, obtained by calling appropriate functions in this list: GetLogicalDevices(..), GetLogicalDeviceHardware(..), GetChannel(..), GetChannelBySite(..), GetPanel(), GetPanelBySite(..) or GetSubpanel(..), GetSubpanelsBySite(..), type of event to be generated, card number, event date. Additional information can be provided in Message field (Optional).
--------------	---

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Events/Alarm types supported

Device Type	Events	Comments
CHANNEL	<i>For issuing Channel events.</i>	
	CHANNEL_COMM_BREAK	
	CHANNEL_STARTED	
	CHANNEL_SUCCESSFULL	
	CHANNEL_FAILED	
	CHANNEL_SIGNON_SUCCESS	
	CHANNEL_SIGNON_FAILED	
	CHANNEL_DISCONNECT_COMPLETE	
PANEL	<i>For issuing Panel events.</i>	
	PANEL_POWER_LOSS	
	RTN_PANEL_POWER_LOSS	Return to Normal
	PANEL_TAMPER	
	RTN_PANEL_TAMPER	Return to Normal
	PANEL_OFFLINE	
	RTN_PANEL_OFFLINE	Return to Normal

	PANEL_RESTART	
	PANEL_LOCAL_ACCESS	
	RTN_PANEL_LOCAL_ACCESS	Return to Normal
	PANEL_BUFFER_FULL	
SUBPANEL	<i>For issuing Subpanel events.</i>	
	SUBPANEL_POWER_LOSS	
	SUBPANEL_BATTERY_LOW	
Category	Events	Comments
	RTN_SUBPANEL_BATTERY_LOW	Return to Normal
	SUBPANEL_TAMPER_ALARM	
	SUBPANEL_OFFLINE	
	SUBPANEL_COMMS_DISABLED	
	SUBPANEL_INVALID_ID	
	SUBPANEL_CMD_LONG	
INPUT POINT	<i>For issuing Input point events.</i>	
	INPUT_ALARM	
	RTN_INPUT_ALARM	Return to Normal
	INPUT_SHORT	
	RTN_INPUT_SHORT	Return to Normal
	INPUT_OPEN	
	RTN_INPUT_OPEN	Return to Normal
	INPUT_HELD_PAST_SHUNT	
	RTN_INPUT_HELD_PAST_SHUNT	Return to Normal
	INPUT_TROUBLE	
	RTN_INPUT_TROUBLE	Return to Normal
	INPUT_MASKED_ENTRY_DELAY	
	INPUT_MASKED_EXIT_DELAY	
	INPUT_FAULT	
	RTN_INPUT_FAULT	Return to Normal
	INPUT_STATUS_UNKNOWN	
	INPUT_DISCONNECTED	
	MONITOR_INPUT_ALARM	
OUTPUT POINT	<i>For issuing Output point events.</i>	
	OUTPUT_ACTIVE	
	RTN_OUTPUT_ACTIVE	Return to Normal
READER	<i>For issuing Reader events.</i>	
	UNKNOWN_CARD	
	VOID_CARD	
	EXPIRED_CARD	
	CARD_TRACE	
	HOST_DENY	
	RESTRICT_READER	
	LOSTCARD_ATTEMPT	
	STOLEN_ATTEMPT	
	UNACCT_ATTEMPT	
	DEACT_ATTEMPT	
	TERMINATED_ATTEMPT	
	INVALID_CARD_TIMEZONE	
	INVALID_READER_TIMEZONE	
	INVALID_PIN	
	INVALID_FACILITY_CODE	
	INVALID_ISSUE	

	INVALID_TO	
	INVALID_INXIT	
	INVALID_THREAT	
	ANTIPASSBACK_VIOLATION	
	RDR_LOCKED_ATTEMPT	
	RDR_USE_LIMIT	
Category	Events	Comments
	RDR_EXIT_GRANTED	
	RDR_EXIT_DENIED	
	RDR_COMM_FAIL	
	RTN_RDR_COMM_FAIL	Return to Normal
	HOST_DENY_VERIFICATION	
	LOCAL_GRANT	
	HOST_GRANT	
	EXEC_PRIV	
	RDR_TIMEDOVERRIDE_ENABLED	
	RDR_TIMEDOVERRIDE_DISABLED	
	RDR_TIMEDOVERRIDE_EXPIRED	
	RDR_UNLOCKED_ATTEMPT	
	LOCAL_GRANT_DURESS_USED	
	LOCAL_GRANT_DOOR_NOT_USED	
	HOST_GRANT_VERIFICATION	
	LOCAL_GRANT_IN_PROGRESS	
	HOST_GRANT_IN_PROGRESS	
	RDR_DISABLED	
	RDR_UNLOCKED	
	RDR_LOCKED	
	HOST_GRANT_USED	

CTRL *	For issuing administrative messages.	
	EVLOG_THRESHOLD_LIMIT	
	DATABASE_ADD	
	DATABASE_UPDATE	
	DATABASE_DELETE	
	DATABASE_QUERY	
	DTU_MESSAGE	Generic event – Use message column to provide more information
	PIVCLASSPAMMESSAGE	FIPS 201 Compliance
	PIVCLASSCERTMGRACTIVATED	FIPS 201 Compliance
	PIVCLASSCERTMGRREVOKED	FIPS 201 Compliance
	PIVCLASSCERTMGRERROR	FIPS 201 Compliance
	PIVCLASSVALIDATIONPASSED	FIPS 201 Compliance
	PIVCLASSVALIDATIONFAILED	FIPS 201 Compliance
	DOWNLOAD_REQUEST	

Below are details on what input fields are required/optional depending on the device type:

Device Type	Issue Event Parameters				
	Hardware ID	Event Type	Event Date	Card Number	Message
CHANNEL	REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL
PANEL	REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL

SUBPANEL	REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL
INPUT POINT	REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL
OUTPUT POINT	REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL
READER	REQUIRED	REQUIRED	REQUIRED	REQUIRED	OPTIONAL
CTRL*	NOT REQUIRED	REQUIRED	REQUIRED	NOT REQUIRED	OPTIONAL

Note: CTRL* events are admin messages and are not associated with any device.

Input Validations:

1. Issue events can only be raised by a valid authorized user in Pro-Watch.
2. An Issue event call (for a reader event) would fail if a card number is not provided.
3. To Issue an event (excluding CTRL events) Hardware ID provided need to be valid in Pro-Watch.
4. The event to be issued should be an event type supported by the addressed device.

An appropriate error message will be returned when any of the above validations fail.

Method: **IssueCtrlEvent** (PwIssueEvent EventMsg)

This method issues a CTRL event in the Pro-Watch system. CTRL events are not tied to any hardware. Please provide event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	---

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: **IssueChannelEvent** (PwIssueEvent EventMsg)

This method issues a Channel event in the Pro-Watch system. Please provide Channel ID (Hardware ID), event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetChannel(..) or GetChannelBySite(..), type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	--

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: **IssuePanelEvent** (PwIssueEvent EventMsg)

This method issues a Panel event in the Pro-Watch system. Please provide Panel ID (Hardware ID), event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetPanel(..) or GetPanelBySite(..), type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	---

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: **IssueSubpanelEvent** (PwIssueEvent EventMsg)

This method issues a Panel event in the Pro-Watch system. Please provide Subpanel ID (Hardware ID), event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetSubpanel(..) or GetSubpanelBySite(..), type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	---

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: **IssueInputEvent** (PwIssueEvent EventMsg)

This method issues a Panel event in the Pro-Watch system. Please provide Input point ID (Hardware ID), event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetLogicalDevices(..) or GetLogicalDeviceHardware(..), type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	--

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: IssueOutputEvent (PwIssueEvent EventMsg)

This method issues a Panel event in the Pro-Watch system. Please provide Output point ID (Hardware ID), event type, event date and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetLogicalDevices(..) or GetLogicalDeviceHardware(..), type of event to be generated and event date. Additional information can be provided in Message field (Optional).
--------------	--

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: IssueReaderEvent (PwIssueEvent EventMsg)

This method issues a Panel event in the Pro-Watch system. Please provide Reader ID (Hardware ID), event type, event date, card number and include any additional informational in the message field to be displayed in message column in Event Viewer/Alarm monitor.

Input Parameters

PwIssueEvent	Necessary fields include Hardware ID, obtained by calling GetLogicalDevices(..) or GetLogicalDeviceHardware(..), type of event to be generated and event date and card number. Additional information can be provided in Message field (Optional).
--------------	--

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Note: Please refer to IssueAnEvent method for additional information.

Method: **QueryBadges** (**PwFilterBadge[]** filters, **PwSortBadge[]** sorts, **bool** getCardData)

Query for a list of badges. Multiple filters can be passed into the method as well as sorting the data. If getCardData is true, card data for each badge is returned

Input Parameters

PwFilterBadge[]	An array of filter data to limit the number of rows returned
PwSortBadge[]	An array of badge fields to sort by
getCardDate	If true, return card data for each badge

Return Value

PwStatusQueryBadges	List of badges return from query as well as the number of records returned
---------------------	--

SampleXML

This call will get all Active cards of badge holders whose last name starts with a 'CA' . The sorting will be by card last change date in a descending order. Clearance codes and door exceptions will not be included in the data.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dtus="http://HoneywellAccess/ProWatch/DTUService"
  xmlns:dat="http://HoneywellAccess/ProWatch/DTUService/DataContracts">
  <soapenv:Header/>
  <soapenv:Body>
    <dtus:QueryCards>
      <dtus:filters>
        <dat:PwFilterCard>
          <dat:FilterCardStatus>Active</dat:FilterCardStatus>
          <dat:FilterField>CardStatus</dat:FilterField>
          <dat:FilterComparer>Equals</dat:FilterComparer>
        </dat:PwFilterCard>
        <dat:PwFilterCard>
          <dat:FilterString>CA</dat:FilterString>
          <dat:FilterField>BadgeLastName</dat:FilterField>
          <dat:FilterComparer>StartsWith</dat:FilterComparer>
        </dat:PwFilterCard>
      </dtus:filters>
      <dtus:sorts>
        <dat:PwSortCard>
          <dat:SortField>LastChangeDate</dat:SortField>
          <dat:SortAscDesc>Descending</dat:SortAscDesc>
        </dat:PwSortCard>
      </dtus:sorts>
      <dtus:includeCardAccess>>false</dtus:includeCardAccess>
    </dtus:QueryCards>
  </soapenv:Body>
</soapenv:Envelope>
```

Method: **QueryBadgesPaging** (int pageSize, int page, PwFilterBadge[] filters, PwSortBadge[] sorts)

Query for a list of badges with the option of setting a page size and page for the returned results. Same function as above QueryBadges with the addition paging option.

Input Parameters

int	The number of badges to be returned
int	The page number of badges to return
PwFilterBadge[]	An array of filter data to limit the number of rows returned
PwSortBadge[]	An array of badge fields to sort by

Return Value

PwStatusQueryBadges	List of badges return from query as well as the number of records returned
---------------------	--

Method: **QueryCards** (PwFilterCard[] filters, PwSortCard[] sorts, bool includeCardAccess)

Query for a list of cards. Multiple filters can be passed into the method as well as sorting the data. To include clearance codes and Door exceptions, set includeCardAccess = true.

Input Parameters

PwFilterCard[]	An array of filter data to limit the number of rows returned
PwSortCard[]	An array of card fields to sort by
boolean	True = include clearance codes and door exceptions for card False = Do not include clearance codes and door exceptions for card

Return Value

PwStatusQueryCards	List of cards return from query as well as the number of records returned
--------------------	---

SampleXML

This call will return all badges that have an Active card. The sorting will be by last name.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dtus="http://HoneywellAccess/ProWatch/DTUService"
  xmlns:dat="http://HoneywellAccess/ProWatch/DTUService/DataContracts">
  <soapenv:Header/>
  <soapenv:Body>
    <dtus:QueryBadges>
      <dtus:filters>
        <dat:PwFilterBadge>
          <dat:FilterCardStatus>Active</dat:FilterCardStatus>
          <dat:FilterField>CardStatus</dat:FilterField>
          <dat:FilterComparer>Equals</dat:FilterComparer>
        </dat:PwFilterBadge>
      </dtus:filters>
    </dtus:QueryBadges>
  </soapenv:Body>
</soapenv:Envelope>
```



```

<dtus:sorts>
  <dat:PwSortBadge>
    <dat:SortField>BadgeLastName</dat:SortField>
    <dat:SortAscDesc>Ascending</dat:SortAscDesc>
  </dat:PwSortBadge>
</dtus:sorts>
</dtus:QueryBadges>
</soapenv:Body>
</soapenv:Envelope>

```

Method: **QueryCardsPaging** (**int** pageSize, **int** page, **PwFilterCard[]** filters, **PwSortCard[]** sorts, **bool** includeCardAccess)

Query for a list of cards. Multiple filters can be passed into the method as well as sorting the data. To include clearance codes and Door exceptions, set includeCardAccess = true.

Input Parameters

pageSize	The number of records to return per call
Page	The page of records to return
PwFilterCard[]	An array of filter data to limit the number of rows returned
PwSortCard[]	An array of card fields to sort by
boolean	True = include clearance codes and door exceptions for card False = Do not include clearance codes and door exceptions for card

Return Value

PwStatusQueryCards	List of cards return from query as well as the number of records returned
--------------------	---

SampleXML

This call will return all badges that have an Active card. The sorting will be by last name.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dtus="http://HoneywellAccess/ProWatch/DTUService"
  xmlns:dat="http://HoneywellAccess/ProWatch/DTUService/DataContracts">
  <soapenv:Header/>
  <soapenv:Body>
    <dtus:QueryBadges>
      <dtus:pageSize>100</dtus:pageSize>
      <dtus:page>1</dtus:page>
      <dtus:filters>
        <dat:PwFilterBadge>
          <dat:FilterCardStatus>Active</dat:FilterCardStatus>
          <dat:FilterField>CardStatus</dat:FilterField>
          <dat:FilterComparer>Equals</dat:FilterComparer>
        </dat:PwFilterBadge>
      </dtus:filters>
      <dtus:sorts>
        <dat:PwSortBadge>
          <dat:SortField>BadgeLastName</dat:SortField>
          <dat:SortAscDesc>Ascending</dat:SortAscDesc>
        </dat:PwSortBadge>
      </dtus:sorts>
    </dtus:QueryBadges>
  </soapenv:Body>
</soapenv:Envelope>

```

```

    </dtus:sorts>
  </dtus:QueryBadges>
</soapenv:Body>
</soapenv:Envelope>

```

Method: **QueryCardsWithBadgePaging** (int pageSize, int page, PwFilterCard[] filters, PwSortCard[] sorts, bool includeCardAccess)

Query for a list of cards. Multiple filters can be passed into the method as well as sorting the data. To include clearance codes and Door exceptions, set includeCardAccess = true.

Input Parameters

pageSize	The number of records to return per call
Page	The page of records to return
PwFilterCard[]	An array of filter data to limit the number of rows returned
PwSortCard[]	An array of card fields to sort by
boolean	True = include clearance codes and door exceptions for card False = Do not include clearance codes and door exceptions for card

Return Value

PwStatusQueryBadges	List of cards and badges return from query as well as the number of records returned
---------------------	--

SampleXML

This call will return all badges that have an Active or Disabled card. The sorting will be by last name.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dtus="http://HoneywellAccess/ProWatch/DTUService"
  xmlns:dat="http://HoneywellAccess/ProWatch/DTUService/DataContracts">
  <soapenv:Header/>
  <soapenv:Body>
    <dtus:QueryCardsWithBadgePaging >
      <dtus:pageSize>100</dtus:pageSize>
      <dtus:page>1</dtus:page>
      <dtus:filters>
        <dat:PwFilterCard>
          <dat:FilterInCardStatus>A,D</dat:FilterCardStatus>
          <dat:FilterField>CardStatus</dat:FilterField>
          <dat:FilterComparer>IsIn</dat:FilterComparer>
        </dat:PwFilterCard>
      </dtus:filters>
      <dtus:sorts>
        <dat:PwSortBadge>
          <dat:SortField>BadgeLastName</dat:SortField>
          <dat:SortAscDesc>Ascending</dat:SortAscDesc>
        </dat:PwSortBadge>
      </dtus:sorts>
    </dtus:QueryCardsWithBadgePaging>
  </soapenv:Body>
</soapenv:Envelope>

```

Method: **SetBadgeBlob** (string badgeID, string blobTypeID, byte[] blob)

This method sets a badge holder photo/signature/electronic document using a Pro-Watch unique badge identifier.

Input Parameters

badgeID	The Pro-Watch badge unique identifier in the form 0x002946374333374146412D333245362D3439
blobTypeID	The Pro-Watch blob type unique identifier in the form 0x00620A011EFD89F94DDA863BA64F57441DE9
Byte[]	The new photo/signature/electronic document

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **SetBadgePhoto** (string badgeID, byte[] photo)

This method sets a badge holder blob/picture using a Pro-Watch unique badge identifier.

This method uses a default Pro-Watch blob types defined in the .config file of this application.

Input Parameters

badgeID	The Pro-Watch badge unique identifier in the form 0x002946374333374146412D333245362D3439
Byte[]	The new photo

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: **SetBadgePhotoByKeyField** (PwCustomBadgeField keyField, byte[] photo)

Set a badge holder blob/picture. A badge key field is used to find the badge holder to update.

This method uses a default Pro-Watch blob types defined in the .config file of this application.

Input Parameters

PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.
Byte[]	The new photo

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: TerminateBadgeCard (string uniqueID)

This method will use the uniqueID to lookup the badge holder in Pro-Watch. If found, the badge status of the badge holder will be set to TERMINATED (TERMINATED must be a badge status), the badge expire date will be set to the Pro-Watch database server current time. Then, for all Active cards, the card status will be set to T (Terminated) and the card expire date will be set to the Pro-Watch database server current time. All clearance codes from all cards (regardless if they were Active or not) will be removed.

Everything is audited in the Pro-Watch event log.

Note: The unique ID badge field column is defined in the .config file of this application, named <IDColumn>. It is defaulted to BADGE_DOMAIN_ID

Input Parameters

uniqueID	This is the custom badge field unique identifier.
----------	---

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: UnlockIfCardAccessLogDev (String LogDevID, String CardNo, String CurrDateTime)

This method will check if an existing card has access to the given door right now and will issue a momentary unlock command on the door.

Parameters

LogDevID	LogicalDevID GUID
CardNo	CardNo from PW Database
CurrDateTime	Date and time with format YYYY-MM-DDThh:mm:ss

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: UpdateAsset (PwAsset asset)

This method will update the description of an asset.

Input Parameters

PwAsset	The asset ID and description to update.
---------	---

Return Value

PwStatus	Pass or Fail.
----------	---------------

Method: UpdateBadge (PwBadge badge)

The Pro-Watch database will be updated if the values passed in are different than what is in the database. Card data sent with this method will try to find the card number to update. If the card is not found, it will assume a new card is being added.

Concurrency: If the PwBadge.RowVersion contains a value when calling UpdateBadge, the Pro-Watch API will check to make sure the current row version in the database matches prior to updating the record. The call will fail if the row versions do not match.

Input Parameters

PwBadge	This is a PwBadge structure that will contain the update data. A valid Pro-Watch unique badge ID must be present for this method to complete successfully.
---------	--

Return Value

PwStatusUpdateBadge	This will return the status of the call and include whether a badge was updated.
---------------------	--

Method: UpdateBadgeAsset (PwBadgeAsset asset)

This method will update an asset assigned to a badge holder. The asset status, various dates and a note can be updated.

Note: Asset statuses are defined in a table valued function called PWAP_KEY_STATUSES. Always send in the text not the code.

Input Parameters

PwBadgeAsset	The asset to be updated for the badge holder.
--------------	---

Return Value

PwStatus	Pass or Fail.
----------	---------------

Method: UpdateBadgeCert (PwBadgeCert certification)

This method will update a certification for a badge holder. Required fields are the badge holder and the row ID of the record to update.

Note: List of attempts comes from database function PWAP_CERTIFICATION_ATTEMPTS. List of results from database function PWAP_CERTIFICATION_RESULTS.

Input Parameters

PwBadgeCert	The certification to be updated for the badge holder.
-------------	---

Return Value

PwStatus	Pass or Fail.
----------	---------------

Method: UpdateBadgeByKeyField (PwBadge badge, PwCustomBadgeField keyField, bool createIfKeyFieldNotFound)

The Pro-Watch database will be updated if the values passed in are different than what is in the database. Card data sent with this method will try to find the card number to update. If the card is not found, it will assume a new card is being added.

If the `createIfKeyFieldNotFound` is set to true, it will first try to find an existing badge holder record using the key field. If the key field is found, the badge holder record will be updated with the new data. If the key is not found, the new data will be added to Pro-Watch as a badge holder.

Input Parameters

PwBadge	This is a PwBadge structure that will contain the update data.
PwCustomBadgeField	This is a custom badge field structure used to identify a Pro-Watch custom badge field and a value used to search for the badge holder.
boolean	If true and if the existing badge holder is not found a new Pro-Watch badge holder record will be created.

Return Value

PwStatusUpdateBadge	This will return the status of the call and include whether a new badge was created or not.
---------------------	---

Method: UpdateCard (PwCard updateCard)

This method will update a card.

Concurrency: If the `PwCard.RowVersion` contains a value when calling `UpdateCard`, the Pro-Watch API will check to make sure the current row version in the database matches prior to updating the record. The call will fail if the row versions do not match.

Input Parameters

PwCard	Card structure to be updated.
--------	-------------------------------

Return Value

PwStatusCard	Returns the status of the card. Card updated, card downloaded.
--------------	--

Method: UpdateCardLogDevEx (PwCardLogDevEx logDevEx)

This method will update a door exception to a card. See `AddCardLogDevEx` for further explanation of door exceptions.

Note: A door exception can be temporarily granted. Set the `TempAccess` flag and set the start and end date/time.

Parameters

PwCardLogDevEx	Parameters used to modify an existing door exception
----------------	--

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: UpdateTimeZone (PwTimeZone pwTimeZone)

This method updates an existing timezone (with schedules) in Pro-Watch. It also issues a timezone download command to the affected panels.

Parameters

PwTimeZone	Timezone (with schedules) to be updated
------------	---

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: UpdateUser (PwUser pwUser)

This method updates a user in Pro-Watch

Parameters

PwUser	User to be updated
--------	--------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Method: UpdateWorkstation (PwWorkstation pwWorkstation)

This method updates a workstation in Pro-Watch

Parameters

PwWorkstation	workstation to be updated
---------------	---------------------------

Return Value

PwStatus	Success or Fail
----------	-----------------

Subscribing to Pro-Watch Events/Alarms

Web DTU's Event Service lets web clients subscribe to real time events from Pro-Watch Service just like a Pro-Watch client using jQuery/JavaScript. The event subscription still obeys routing groups and partitions, i.e. a web user can only receive events that the user's routing groups allows it and if it is visible to the user based on the partitions assigned to the user.

The "**Subscribe to Events**" permission should be assigned to a Pro-Watch user to give the user the ability to perform this action. To add the "Subscribe to Events" permission for a Pro-Watch user, go to the Program tab under User's Properties, select **Programs->Administration->Data Transfer Utility** and click on "**Add Function...**" button and select "**Subscribe to Events**" function to assign the permission to the user.

The Event Service uses SignalR (part of ASP.NET Web API framework) a library to push new data (events) at real-time to connected clients instead of waiting for clients to request new data. Depending on the client and server side support available [refer to <http://www.asp.net/signalr/overview/getting-started/supported-platforms>], SignalR can use webSocket, EventSource, Forever Frame or Ajax long polling for communication. Visit [<http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr>] for more information.

Subscribing to Pro-Watch Events using JavaScript Client

The JavaScript client requires references to jQuery and SignalR core JavaScript file. The jQuery version must be 1.6.4 or major later versions. The following example shows what the references look like in an HTML page:

```
<script src="Scripts/jquery-2.1.3.min.js" ></script>
<script src="Scripts/jquery.signalR-2.2.0.min.js"></script>
```

Note: These references must be included in order: jQuery first and the SignalR core after that.

The DTU event service runs at the following default URL: <http://localhost:8735/pwevents>. This information can be located in the application's configuration file and can be modified.

Here are the following steps to establish connection with the service and start to receive events:

1. Establish connection with the DTU event service using the *hubConnection* object:

```
var conn = $.hubConnection("http://localhost:8735/pwevents", {
    useDefaultPath: false });
```

2. Create a proxy to call the server side code:

```
var myhub = conn.createHubProxy("PWEventService");
```

3. Set username and workstation name (created in Pro-Watch) for the connection. This needs to be done before you subscribe to events. This information is required by the event service to route only the relevant events to the client.

```
myhub.state.userName = <username>;
myhub.state.wrkstName = <workstation name>;
```

4. Before establishing a connection, you have to create a connection object, proxy, and register event handlers for methods that can be called from the server.

```
myhub.on("onProwatchEvent", function (pwEvent) { ... }
myhub.on("onProwatchAlarm", function (pwEvent) { ... }
myhub.on("onProwatchAlarmDisposition", function (pwEventDisp) { ... }
```


- When the proxy and event handlers are setup, establish the connection by calling start method and on completion, subscribe to the Event Service to receive events/alarms.

```
conn.start().done(function () {
...
myhub.invoke('subscribe');
...
});
```

- To stop receiving events and alarms, call the unsubscribe method

```
myhub.invoke('unsubscribe');
```

Note: Make sure the URL of the web client (using SignalR Event service) is included in the value of [CorsOriginSettings](#) key in API configuration file. Failing to do so will cause the connection request to be rejected.

Subscribing to Pro-Watch Events using C# Client

A reference to Microsoft.AspNet.SignalR.Client needs to be added to the client project. The steps to connect to and subscribe to the event service are very similar to those taken for a JavaScript client.

- Establish connection with Event service.

```
HubConnection conn = new HubConnections(url);
```

- Create a proxy to call the server side code:

```
IHubProxy EventSrvProxy = conn.CreateHubProxy("PWEventService");
```

- Before establishing a connection, you have to create a connection object, proxy, and register event handlers for methods that can be called from the server.

```
EventSrvProxy.On<PwEvent>("onProwatchEvent", OnProwatchEvent);
EventSrvProxy.On<PwEvent>("onProwatchAlarm", OnProwatchAlarm);
EventSrvProxy.On<PwEventDisposition>("onProwatchAlarmDisposition",
OnProwatchAlarmDisposition);
```

- When the proxy and event handlers are setup, establish the connection by calling start method and on completion, subscribe to the Event Service to receive events/alarms.

```
Conn.Start().Wait();
PwStatus status = EventSrvProxy.Invoke<PwStatus>("subscribe").Result;
```

- To stop receiving events and alarms, call the unsubscribe method

```
EventSrvProxy.Invoke("unsubscribe");
```

Server Methods

Event Subscription – (Using Proxy)

Method: `subscribe ()`

This method is used to subscribe to events and alarms from Pro-Watch. The username and workstation name already been provided should be a valid Pro-Watch entity, the user should have permissions to subscribe to events and the workstation should be in the list of workstations that the Pro-Watch user has access to.

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Method: `unsubscribe ()`

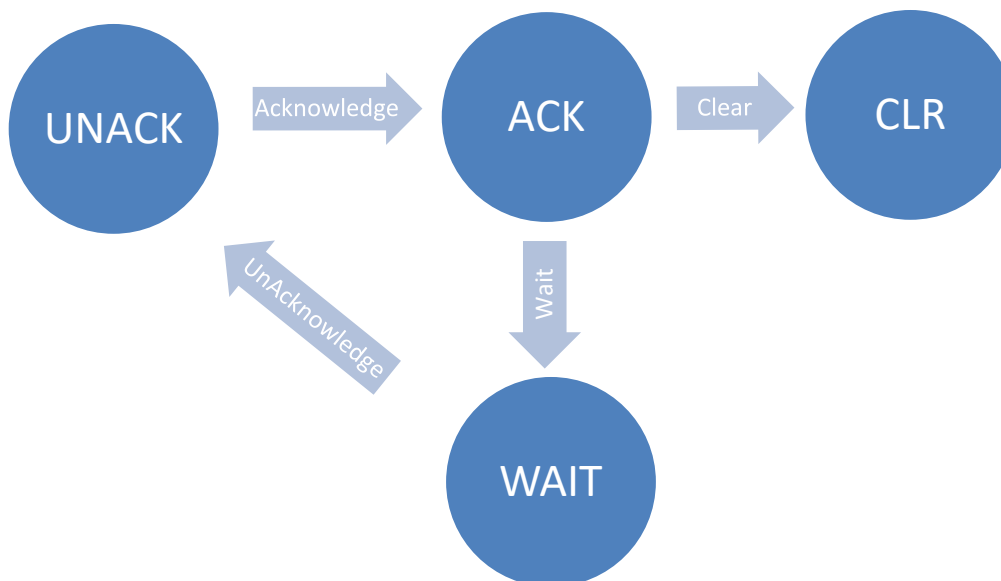
This method is used to unsubscribe to events and alarms from Pro-Watch.

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Alarm startup and actions – (Using Rest API)

This section describes API's that will help in getting all unacknowledged events in the system, the latest dispositions for all alarms in the system and actions that can be performed on the alarms.

Alarm State transitions (Dispositions)

Http Request header for the Rest API calls should contain the following information

Authorization: Basic <Base64 Encoded value of (username:password)>

content-type: application/json

X-PW-WRKST: <workstation name>

Action	Resource	Description
GET	/alarms	Gets all alarms in system that have not been cleared
GET	/alarms/state	Gets current dispositions for all alarms in the system
PUT	/alarms/{eventId}/state/acknowledge	Acknowledge an alarm with ID {eventId}.
PUT	/alarms/{eventId}/state/clear	Clear an alarm with ID {eventId}.
PUT	/alarms/{eventId}/state/wait	Put an alarm with ID {eventId} in wait state. The body of request contains wait state parameter. To put an alarm to Wait Indefinitely: <pre>{ "WaitIndefinitelyFlag": 1 }</pre> <i>To put an alarm to Wait for certain amount of time (in min):</i> <pre>{ "WaitIndefinitelyFlag": 0, "WaitTime": 5 }</pre> Note: When wait time expires the alarm is put in unacknowledged state.
PUT	/alarms/{eventId}/state/unacknowledge	Put an alarm with ID {eventId} in unacknowledged state.

Client Methods

Method: `onProwatchEvent` (`PwEvent` pwEvent)

When the Event service receives a new event from Pro-Watch service it calls the event handler to this event, registered by the client, to send the event to subscribed clients. The client will only receive the event if the routing group and partition allows it.

Input Parameters

PwEvent	This structure contains all the information about the event issued by Pro-Watch service.
---------	--

Method: `onProwatchAlarm` (`PwEvent` pwEvent)

When the Event service receives a new alarm from Pro-Watch service it calls the event handler to this event, registered by the client, to send the alarm to subscribed clients. The client will only receive the alarm if the routing group and partition allows it.

Input Parameters

PwEvent	This structure contains all the information about the alarm issued by Pro-Watch service.
---------	--

Method: `onProwatchAlarmDisposition` (`PwEventDisposition` pwEvent)

When the Event service receives a new alarm disposition (change of alarm state) from Pro-Watch service it calls the event handler to this event, registered by the client, to send the alarm disposition to subscribed clients. The client will only receive the alarm if the routing group and partition allows it.

Input Parameters

PwEventDisposition	This structure contains all the information about the alarm disposition issued by Pro-Watch service.
--------------------	--

Subscribing to Pro-Watch Data Change Events

The ProWatch API Data Service lets clients subscribe to real time data change events from the Pro-Watch database. A client can only receive data change notifications for “published” tables (see how to publish tables later in this documentation).

The Data Service uses SignalR (part of ASP.NET Web API framework) a library to push new data (events) at real-time to connected clients instead of waiting for clients to request new data. Depending on the client and server side support available [refer to <http://www.asp.net/signalr/overview/getting-started/supported-platforms>], SignalR can use WebSocket, EventSource, Forever Frame or Ajax long polling for communication. Visit [<http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr>] for more information.

Note: To Activate the ProWatch data service it must be enabled in the ProWatch API Configuration File. The following entries should be configured and enabled:

```
<!-- Data Service Url , replace the localhost with the name of the computer running API -->
<add key="PWDataSignalRUrl" value="http://localhost:8736/" />

<!-- Start the Pro-Watch Data Service -->
<add key="StartDataService" value="1" />
```

Required Database Services and Permissions

To use notifications, you must be sure to enable the Microsoft SQL Service Broker for the database. To do this run the SQL command below (where MyDatabase is the name of the ProWatch database typically PWNT): Please close all client connections prior to running the command below to avoid any Timeout issues. Also ensure the logged in user has permission to enable the service broker

```
ALTER DATABASE MyDatabase SET ENABLE_BROKER
```

In case the user running the ProWatch API Service is not a database **Administrator, db owner** or has the **db_owner** role, make sure to GRANT the following permissions to that SQL user:

- ALTER
- CONNECT
- CONTROL
- CREATE CONTRACT
- CREATE MESSAGE TYPE
- CREATE PROCEDURE
- CREATE QUEUE
- CREATE SERVICE
- EXECUTE

- SELECT
- SUBSCRIBE QUERY NOTIFICATIONS
- VIEW DATABASE STATE
- VIEW DEFINITION

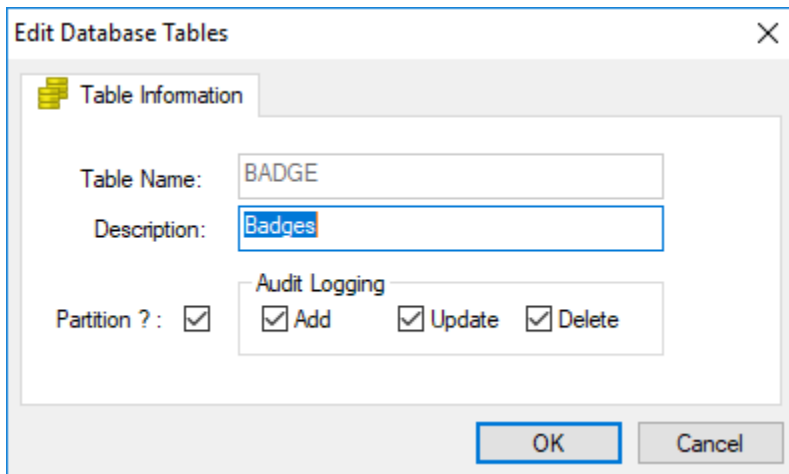
ProWatch Data Service Configuration

The **"Subscribe to Data"** permission should be assigned to a Pro-Watch user to give the user the ability to perform this action. To add the "Subscribe to Data" permission for a Pro-Watch user, go to the Program tab under User's Properties, select **Programs->Administration->Data Transfer Utility** and click on **"Add Function..."** button and select **"Subscribe to Data"** function to assign the permission to the user.

The ProWatch Data Service monitors the ProWatch Audit Log table for changes. The ProWatch Audit Log table is updated anytime a change occurs to any "audited" ProWatch table (note: not all tables are marked as being audited). The ProWatch data service requires both the table(s) to be marked as being audited, and being published. to be transmitted to the SignalR client(s).

Enabling a prowatch table to be audited:

Start the ProWatch Administration Application and navigate to "Database Configuration => Database Tables => (table you are interested in monitoring)". For example, clicking on the "Badges" table icon brings up the Table Information Dialog as seen below. Select the "Audit Logging – Add, Update, Delete" checkboxes" to enable those operations to be audited, and records to be added to the Audit Log anytime an add/update or delete operation on a badge occurs.



The screenshot shows a dialog box titled "Edit Database Tables". It has a tab labeled "Table Information". Inside the dialog, there are two text input fields: "Table Name:" with the value "BADGE" and "Description:" with the value "Badges". Below these, there is a section for "Audit Logging" containing three checkboxes: "Add", "Update", and "Delete", all of which are checked. To the left of this section is a checkbox labeled "Partition ?" which is also checked. At the bottom right of the dialog are "OK" and "Cancel" buttons.

Enabling a ProWatch Table to be Published:

Currently a "Publish" check box does not exist in the ProWatch interface above, however the ProWatch "TABLES_ACCESS" table which contains the values for the "Add/Update/Delete" checkboxes contains a column called "Publish". Setting a value of 1 in this column lets the system know you are interested in Publishing that table change information to the Data Service. For example, to Publish all the Badging Tables you will need to turn

on Auditing as in the example above. Then enter the following SQL Command inside Microsoft SQL Query Manager to activate publishing for the badging changes.

```
UPDATE TABLES_ACCESS SET PUBLISH = 1 WHERE TABNAME IN ('BADGE', 'BADGE_V', 'BADGE_C', 'BADGE_CC', 'BADGE_CERT', 'BADGE_CL', 'BADGE_K', 'BADGE_SCC', 'BLOBS')
```

Subscribing to Pro-Watch Events using C# Client

A reference to Microsoft.AspNet.SignalR.Client needs to be added to the client project. The steps to connect to and subscribe to the event service are very similar to those taken for a JavaScript client.

1. Establish connection with Data service.

```
var connection = new
HubConnectionBuilder().WithUrl("http://localhost:8736/PWDataService").WithAutomaticReconnect().Build();

connection.StartAsync().ConfigureAwait(false).GetAwaiter().GetResult();
```

2. Register Data Change Notification handlers for methods that can be called from the server.

```
connection.On<List<AuditLog>>("OnAuditLogDataChange",
OnAuditLogDataChange);
```

```
var status = connection.InvokeAsync<PwStatus>("Subscribe",
filter).ConfigureAwait(false).GetAwaiter().GetResult();
```

3. Create a Data Service Subscription Filter (contains the ProWatch user connecting to the server, and an Entities Filter which currently only supports "AuditLog"):

```
var entity = new[] { "AuditLog" };
var filter = new DataServiceSubscriptionFilter("Put PW User Name Here",
entity)
```

4. To start receiving Data Change events, call the Subscribe method

```
var status = connection.InvokeAsync<PwStatus>("Subscribe",
filter).ConfigureAwait(false).GetAwaiter().GetResult();
```

Server Methods

Method: **Subscribe** ()

This method is used to subscribe to Data Change Events from Pro-Watch. The username that has already been provided should be a valid Pro-Watch entity, the user should have permissions to subscribe to data change events.

Return Value

PwStatus	The status of the operation
----------	-----------------------------

Client Methods

Method: **OnAuditLogDataChange** (List<AuditLog> log)

When the Data service receives a new data change event from Pro-Watch it calls the event handler to this event, registered by the client, to send the event to subscribed clients.

Input Parameters

List<AuditLog> log	<p>This structure contains all the information about the data change event issued by Pro-Watch including before and after values for each change. Fields in the structure are:</p> <p>BatchID : Unique ID for a batch of changes</p> <p>TableName : The name of the ProWatch Table that had the change</p> <p>ColumnName : The name of the column that changed</p> <p>AuditTime : The date and time the change occurred</p> <p>Operation : The type of operation Add, Update, Delete</p> <p>Key1 : Primary Key of the TableName</p> <p>Key2 : additional Primary Key of the Table (if multiple)</p> <p>Key3 : additional Primary Key of the Table (if multiple)</p> <p>BeforeImage : The value before the change</p> <p>AfterImage : The value after the change.</p>
--------------------	--

HTTPS/SSL support in Pro-Watch DTU Service

HTTPS is used for secure communication over the internet. Simply put, HTTPS uses HTTP protocol on top of SSL protocol thus adding security capabilities of SSL to standard HTTP communications. Here are the following steps to add HTTPS/SSL support in Pro-Watch DTU service:

1. **Creating SSL Certificates (For testing purposes only):** SSL certificates are the most essential part of establishing an SSL connection. A SSL certificate is required by server to prove its identity to the client. To create a self signed certificate (for testing purpose) you could run the makecert.exe utility located in Microsoft Sdk's Bin folder depending on the version of SDK installed (Example: *C:\Program Files\Microsoft SDKs\Windows\v6.0A\Bin*).

Creating Root Certificate

```
makecert.exe -sk RootCA -sky signature -pe -n CN=<machine name> -r -sr LocalMachine -ss Root
<root certificate name>.cer
```

Creating Personal Certificate

```
makecert.exe -sk server -sky exchange -pe -n CN=<machine name> -ir LocalMachine -is Root -ic <root
certificate name>.cer -sr LocalMachine -ss My <personal certificate name>.cer
```

Note: An alternate way to create a certificate is in IIS

You could view the created certificates through MMC snap-in. Please refer to this MSDN link for additional information: <http://msdn.microsoft.com/en-us/library/ms788967.aspx>. Here is a screen capture of Root certificate and Personal certificate as shown in MMC.

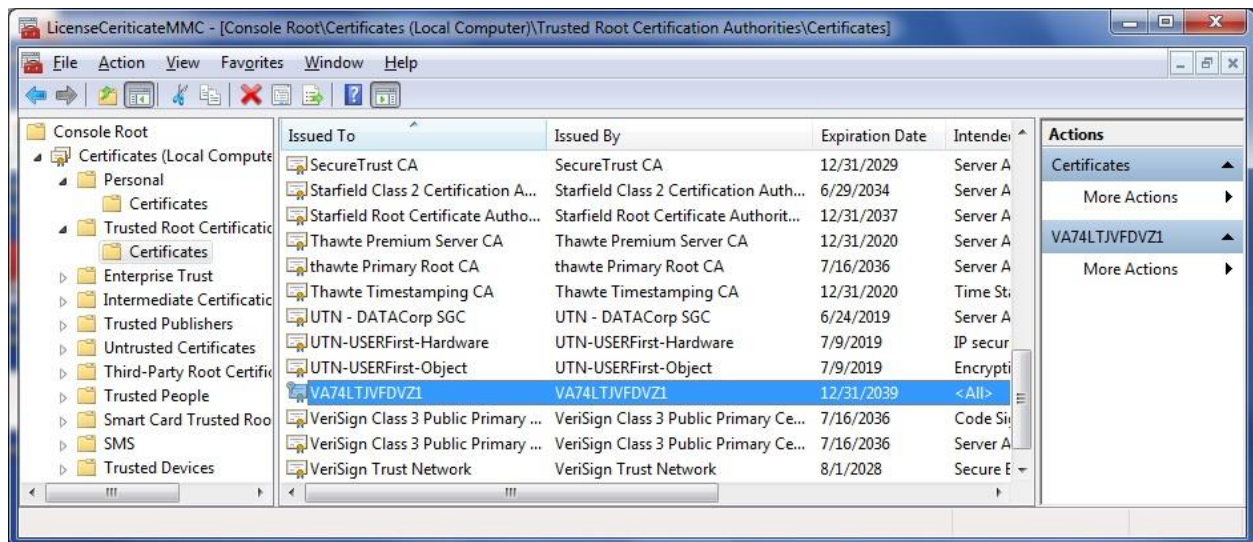


Figure: Root certificate created for Machine name: VA74LTJVFVDVZ1 as seen on MMC

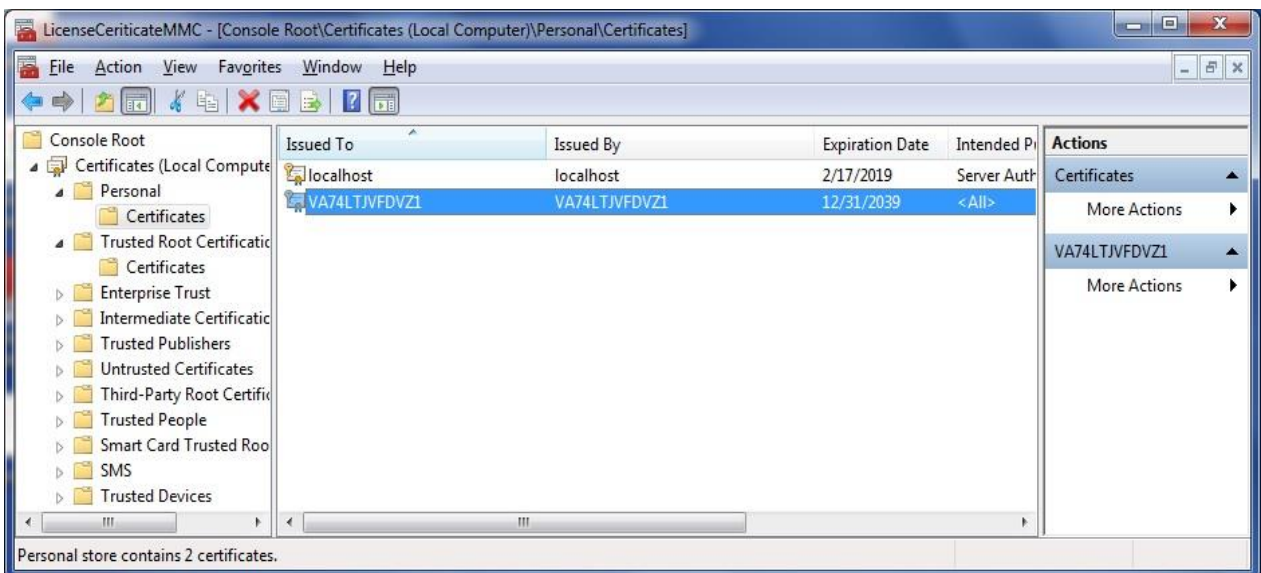


Figure: Personal certificate created for Machine name: VA74LTJVFVDVZ1 as seen on MMC

2. **Binding SSL certificates:** The next step is to bind the SSL certificate (personal) to the port that hosts DTU service. This allows the DTU service to take advantage of SSL and support HTTPS communication. The "netsh" command is used to bind the certificate to the port. The parameters for this command include port number, thumbprint of the certificate and the GUID of the application/service that is authorized to use this certificate. Additional information about the command can be found here: <http://msdn.microsoft.com/en-us/library/ms733791.aspx>. The thumbprint for the certificate can be found using the MMC. Please follow the link <http://msdn.microsoft.com/en-us/library/ms734695.aspx> for additional information.

netsh http add sslcert ipport=0.0.0.0:<port number> certhash=<thumb print> appid={<Application GUID>}

DTU Service SOAP GUID: {8f1e1264-27b8-4399-806a-1b36b6e4e48e}

DTU Service REST GUID: {8f6bfb6e-b790-4682-b0c0-b9196e6eaebc}

***SOAP example: netsh http add sslcert ipport=164.145.215.243:8732
certhash=b327763fb4777ffd5fb3395460 8e41290bb39516 appid={8f1e1264-27b8-4399-806a-1b36b6e4e48e}***

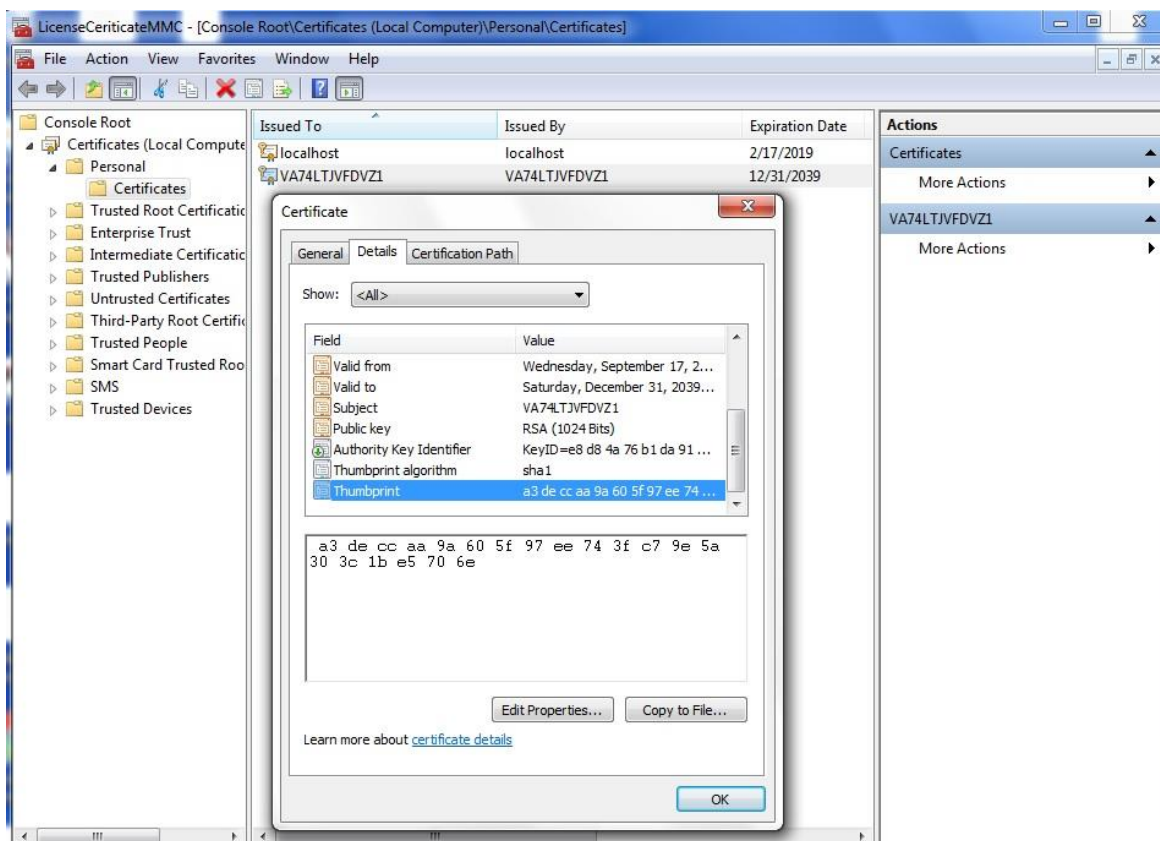


Figure: SSL certificate thumbprint

3. **Service Configuration changes:** In order to support HTTPS communication the following changes need to be made to the Pro-Watch DTU service XML configuration file:
 - a) Change the baseAddress to use https protocol instead of http.
`<add baseAddress="https://localhost:8732/ProWatch/DTUService"/>`
 - b) Change the binding for the endpoint (with name = "basicHttpBinding") to "wsHttpBinding" from "basicHttpBinding". Update the name to "wsHttpBinding" as well. Also change the bindingConfiguration value to "ProWatch_DTU_WS_Http_Binding"
 - c) Change the binding for the endpoint (with address = "mex") to "mexHttpsBinding" from "mexHttpBinding".
 - d) Under the <bindings> tag, add <wsHttpBinding> similar to <basicHttpBinding> tag.

```

<wsHttpBinding>
  <binding name="ProWatch_DTU_WS_Http_Binding">
    <security mode="Transport">
      <transport clientCredentialType="Basic"/>
    </security>
  </binding>
</wsHttpBinding>

```

- e) In the <behaviors> section, change the "httpGetEnabled" to "httpsGetEnabled"
- f) After starting Pro-Watch DTU service, you can now access the service URL (HTTPS) to obtain the WSDL file. If it is using a self-signed certificate, like the one created in Step 1, it will not be on the browsers trusted list of certificates. Select the option to "continue" with the "un trusted certificate" in order to access the URL. (For testing purposes only)

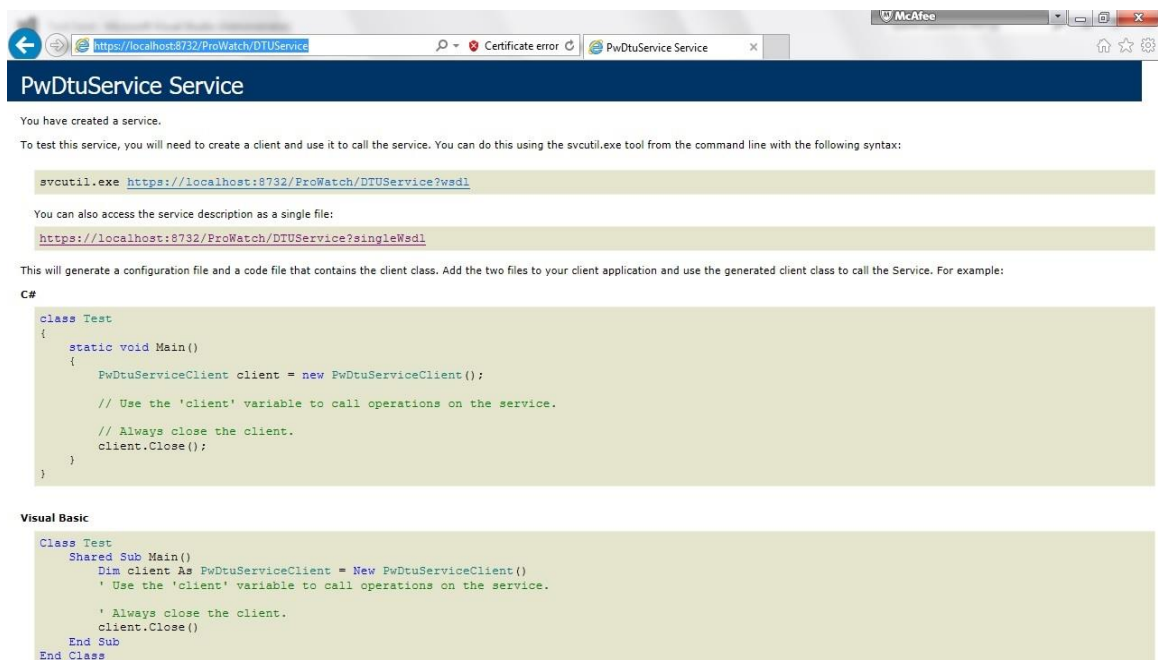


Figure: Pro-Watch DTU service URL using HTTPS protocol

Troubleshooting Note: If browsing to <https://localhost:8732/prowatch/dtuserice> and the DTU Service does not display, the Windows Application Log may help troubleshoot.

4. **Client Configuration changes (For testing purposes only):** The client XML configuration file could be modified to accept an un-trusted self signed certificate from the server. Code changes at the client could also be made to not validate the certificates. This is unsafe and should be done only in test environments. One could add the certificate (to be trusted) to XML config file or in the code instead of accepting all certificates.

Add <behavior> under <endpointBehaviors> as follows to remove certificate validation:

```

<behaviors>
  <endpointBehaviors>

```

....

```
<behavior name="DisableServiceCertificateValidation">
  <clientCredentials>
    <serviceCertificate>
      <authentication certificateValidationMode="None"
        revocationMode="NoCheck" />
    </serviceCertificate>
  </clientCredentials>
</behavior>
```

You could also add the following code in the client to stop certificate validation:

```
System.Net.ServicePointManager.ServerCertificateValidationCallback = ((sender, certificate, chain,
sslPolicyErrors) => true)
```