RS2
REST API v7.0.0.x
User's Guide

Last Update: 2/27/2019 4:29PM

Release Notes

- Web API version increased to 7.0.0.x to coincide with the version of Access It!

Requirements

- HTTPS is now required to use the Web API.
- The Web API must be activated on the Access It dongle.
- A client key must be entered into the system, using either the thick client or the web client, in order to activate the Web API. (Provided by RS2)
- A seperate Public Key must be passed with all calls to the API for the call to work. (Provided by RS2)

Introduction

The RS2 REST API can be used for the following purposes:

- Fetch system data including hardware settings and events.
- Update system data such as reader and card settings.
- Execute commands on hardware such as unlocking a door or executing a macro.

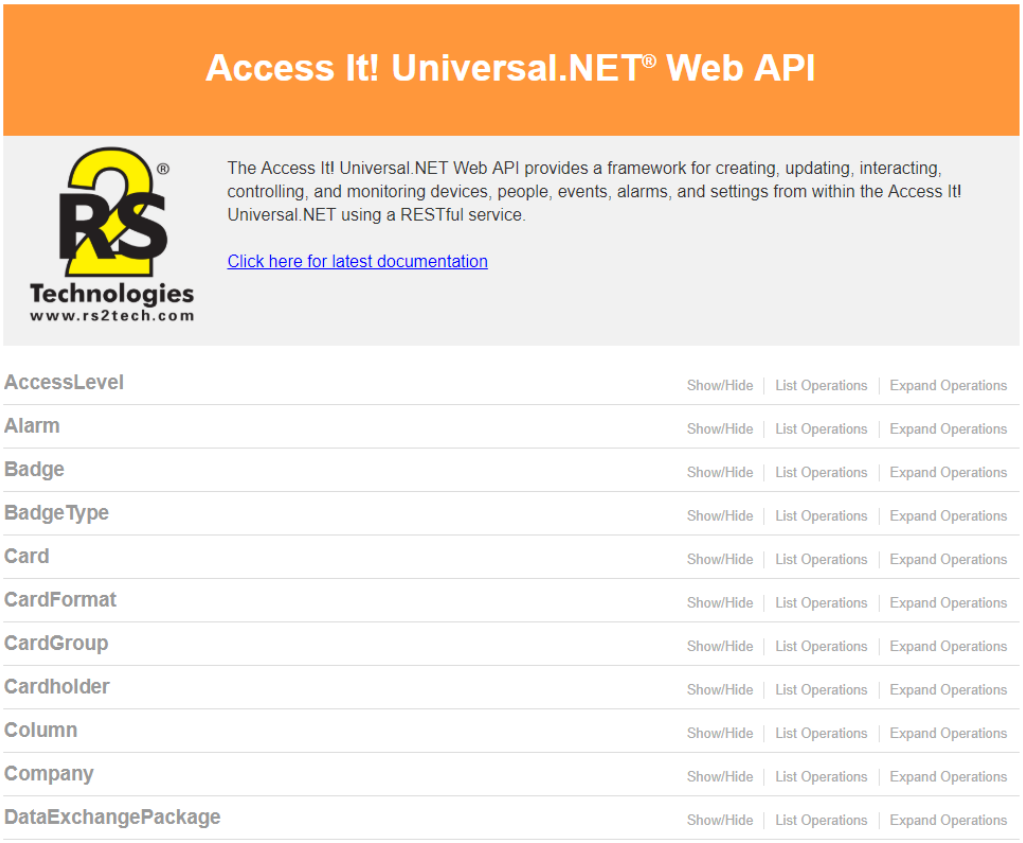
There is comprehensive online documentation available and that should be the first point of reference for a developer looking to integrate with the API. This document is intended to provide additional notes and information that could not easily be included in the online documentation.

If the Web API feature is enabled and the Access It! Universal.NET service is running then the online documentation will be available at the following URL:

http:///(Access It! Universal.NET Server):(Port)/swagger/ui/index

Note that the Web API port is 55459 by default but can be set to a different number within the Access It! Universal software from the “Edit Server” screen.

Accessing the above URL will provide a list of available API calls as shown below:



Clicking on each item in the list will reveal more details about the API calls available for each area of the API (Alarm, Badge, etc.). It is possible to send GET/POST/PUT commands directly using the above web interface. Basic authentication is used for all calls and we strongly recommend using SSL (https) in production. A valid Access It! Universal.NET username/password must be supplied when executing each command. A permission check will be applied to each REST call according to the username provided.

Swagger/OpenAPI and Client Code Generation

Swagger (also known as OpenAPI) is an API description format for REST APIs. It is analogous to the WSDL document format that is often used to describe a SOAP-based web service. For more information see https://swagger.io/docs/specification/about/.

The Swagger/OpenAPI description document (in JSON format) for RS2's REST API can be viewed here:

http:///(Access It! Universal.NET Server):(Port)/swagger/docs/v1.0

This JSON document can easily be imported into the Postman application (see: http://www.getpostman.com) if desired. Postman provides another way to quickly try out the API.

The Swagger description document can also be used to generate client code in many languages using Swagger-CodeGen (see: https://swagger.io/tools/swagger-codegen/). Alternatively, a developer can write their own custom code to execute HTTP GET/PUT/POST commands if preferred.

Abbreviations/Glossary of Terms

The reader can learn more about Access It! Universal.NET by using the help documentation that is included with that software (accessed using the File -> help menu item). This will help with understanding basic terminology such as “Timezone”, “Interval”, “Precision Access”, and “Access Level”. The reader should also be aware of the following abbreviations and/or terms that are used within the API:

Abbreviation	Meaning
SCP	System Control Processor board (main board)
SIO	System Input/Output board (typically used to provide extra inputs/outputs/readers)
CP (e.g. CPNumber)	Output (e.g. Output Number)
ACR (e.g. ACRNumber)	Reader (e.g. Reader Number)
MP (e.g. MPNumber)	Input (e.g. Input Number)
Ack (e.g. SecureAck)	Acknowledgement (e.g. Secure Acknowledgement)

API Notes

General

GET (lists) versus GET (single item)

There are often two different GET requests per item - one that returns a list of items and another that returns a single item. For some items the GET request that returns a list (e.g. list of cards) does not return as much item data as the GET request for a single item (e.g. a single card). This

is by design in order to improve performance for the list request. If full information is needed on each item then it can be fetched using the item-specific GET for each item.

PUT

When creating a new item such as a new card (using the POST operation) the data for the new item should be passed in the body of the HTTP request. The following data can be omitted:

- The primary key for the item. This can be set to null or the empty GUID. For example it is not necessary to provide a CardID when creating a new card. The primary key will be generated by the system.
- Last modified (date). This will be generated by the system.
- Last modified by User (string). This will be generated by the system.

TODO: Check for any Enums that aren't already described and document them here.

TODO: Go through all of the controllers and document any unusual/unexpected business rules or behavior.

Alarms

PUT /v1.0/Alarms/{id}

This call is used to either acknowledge or clear an alarm.

If the alarm has not yet been acknowledged then it will be acknowledged. If the alarm has already been acknowledged then it will be cleared.

Badges

GET /v1.0/Badges

Filter Parameter

The optional "filter" querystring parameter is used to filter down the results and is based on the oData filter (see: http://www.odata.org/documentation/odata-version-2-0/uri-conventions/ section 4.5).

Examples
filter=cardnumber eq 123
filter=cardnumber ne 55
filter = lastname eq smith
filter=startswith(lastname, 'jo')

filter=startswith(lastname, jo) (note that single quotes are optional around strings)
filter=cardnumber eq 123 and startswith(lastname, smit)
filter=(cardnumber eq 123) and (startswith(lastname, smit))
filter=(cardnumber eq 123) or (startswith(lastname, smit))
filter=cardnumber ge 1000
filter=datecreated gt datetime'2018-06-19T01:30:00'

Notes on data values:

- Strings can be enclosed in single quotes if desired but this is not required.
- A datetime value should be written as follows: datetime'yyyy-mm-ddThh:mm:ss'. For example: datetime'2018-07-04T09:00:00' which represents 9AM on July 4th 2018 local time. Only local times are currently supported, it is not possible to specify a timezone.

Allowed filter operators (note that operators are not case sensitive in the URL):
String operators:

StartsWith(fieldname, string)
SubstringOf(fieldname, string)
EndsWith(fieldname, string)

Comparison operators (can be used to filter numeric/datetime/character fields):

Gt (greater than)
Lt (less than)
Eq (equal)
Ne (not equal)
Ge (greater than or equal to)
Le (less than or equal to)

Logical operators (used to apply multiple statements):

Or (logical OR)
And (logical AND)

Parentheses ( ) can be used to group logical statements together, as shown in the examples above.

OrderBy Parameter

Examples:

?orderby=LastName,FirstName
?orderby=DateCreated,LastName,FirstName
?orderby=CardNumber

The items will be returned sorted on the specified fields, in ascending order.

LastCardTimestamp and LastCardholderTimestamp Parameters

NOTE: when one or the other of these parameters is specified the other parameters in the query string will be ignored.

The LastCardTimestamp and LastCardholderTimestamp parameters can be used to only fetch badges where the card or cardholder has been updated since the provided timestamps. The timestamp parameter is a base-64 encoding of a row version number. Row row version numbers are increased each time a row is changed.
(Note that despite the name, Timestamp parameters do not contain date or time information.)

Example timestamp: AAAAAAAD79U= which when base-64 decoded is 0x000000000003EFD5

See the information below on the GET /v1.0/Badges/LastModification call for details on how to fetch the most recent time stamps for cards and cardholders.

GET /v1.0/Badges/LastModification

This call returns LastCardTimestamp and LastCardholderTimestamp values which can then be passed as querystring parameters to the GET /v1.0/Badges call. This allows the client to only return badges that were added or updated after the last card/cardholder timestamp (i.e. newly added or modified records only).

Cards

GET /v1.0/Cards

The following sets of enumerated values are referenced in the card data:

IPLocksetUserType enum:

- 1 - Master User
- 2 - Emergency User
- 3 - Deadbolt Override
- 4 - Regular User
- 5 - Extended User
- 6 - Passage User
- 7 - Use Limit User
- 8 - Panic User
- 9 - Lockout User
- 10 - Relock User
- 11 - Notify User
- 12 - Comm User
- 13 - Suspended User

IPLocksetAccessMode enum:

1 - Card Only
2 - Card Or PIN
3 - Card And PIN
4 - Card Then PIN

POST v1.0/Cards/{id} and PUT v1.0/Cards{id}

When adding or updating a card the following fields can be omitted. They will be generated by the system.

In the CardAccessLevels array:

CardAccessLevelId

In the PrecisionDetails array:

PrecisionDetailID

In the IPLocksetPrecisionDetails array:

IPLocksetPrecisionDetailID

In the CardEvents array:

CardEventID

Card Event Reporting - CardEvents with PUT/POST

The card events array is used to customize reporting on card events for the various event types. If an EventType is omitted from the CardEvents array then event reporting settings will revert to the default setting for that event type. Setting the CardEvents array to an empty array will cause the settings for all event types to be reset to the system defaults.

Note that any customized card events settings will only take effect if the “UseCustomReporting” field is set to true.

Cardholders

POST v1.0/Cardholders/{id} and PUT v1.0/Cards/{id}

When adding or updating a cardholder the following fields can be omitted. They will be generated by the system.

In the CardholderAccessLevels array:
    CardholderAccessLevelId

TODO: Any extra information needed for child data such as AccessLevels?

Columns

GET /v1.0/Columns/{tableName}

Currently the only supported table names are cards and cardholders.

Messages - Events, Alarms and Change of State Notifications

Polling

Messages can be polled using the /Messages REST API call. See the online documentation for more information.

Messages are cached in the system for 60 seconds and each message has a unique sequence number. Sequence numbers are incremented by one for each new message that arrives. The /v1.0/Messages/{startingSequenceNumber} GET call can be used to fetch all messages starting from a given sequence number.

SignalR

Messages can be received by a client using Microsoft's SignalR technology (see https://www.asp.net/signalr). The messages are received approximately in real time. The message data is as described in the "Message Types" section above.

The URL for the SignalR hub is:

http:///(Access It! Universal.NET Server):(Port)/signalr

If an SSL certificate is installed then https should be used instead of http.

The name of the hub for use in a client proxy is “MessageHub”.

Message Data

There are 6 different message types that can be returned. These are listed below including data fields and data types. Lists of enumerated values (“enums”) that are used to set certain message fields are also provided.

InputState

SequenceNumber (int)
MessageType = MessageTypes.InputChangeOfState
InputId (Guid)
State (InputStates enum) (Byte)
InputInAlarm (bool)
MaskState (MaskStates enum)

MessageTypes enum:

```
Transaction = 0,
Alarm = 1,
ReaderChangeOfState = 2,
InputChangeOfState = 3,
OutputChangeOfState = 4,
IpLocksetChangeOfState = 5
```

InputStates enum:

```
isUnknown = 0
isUnsecure = 1
isSecure = 2
isFault = 3
```

OutputState

SequenceNumber (int)
MessageType = MessageTypes.OutputChangeOfState
OutputId (Guid)
State (OutputStates enum)
OutputInAlarm (boolean)

OutputStates enum:

osUnknown = 0
osActive
osInactive

Transaction (Event)

SequenceNumber (int)
MessageType = MessageTypes.Transaction;
SiteId (Guid)
SiteName (string)
EventDate (datetime)
SourceType (SourceTypes enum)
SourceId = (Guid)
EventType = (Event types enum)
Description = (string)
Location = (string)
Cardholder = (string)
CardNumber = (long)
FacilityCode = (int)

SourceTypes enum:

stUndefined = 0
stSystem '1
stSCP '2
stSIO '3
stMP '4
stCP '5
stAcr '6
stTimeZone '7
stProcedure '8
stTrigger '9
stArea '10
stOperator '11

EventTypes enum:

The /Alarms/{id} REST API call online documentation can be used to view a complete list of EventTypes enum values. In the response section, click on “Model”.

ReaderState

SequenceNumber = sequenceNumber;
MessageType = MessageTypes.ReaderChangeOfState

ReaderId (Guid)
CurrentReaderMode (ReaderModes enum)
DoorState (DoorStates enum)
DoorInAlarm = (boolean)
ForcedOpenState (DoorAlarmStates enum)
ForcedOpenMaskState (MaskStates enum)
HeldOpenState = (DoorAlarmStates enum)
HeldOpenMaskState = (MaskStates enum)
AccessCycleState = (AccessCycleStates enum)

ReaderModes enum:
rmUnknown = 0
rmDisabled = 1
rmUnlocked = 2
rmLocked = 3
rmFacilityCodeOnly = 4
rmCardOnly = 5
rmPINOnly = 6
rmCardAndPIN = 7
rmCardOrPIN = 8

DoorStates enum:
dsUnknown = 0
dsOpen = 1
dsClosed = 2
dsFault = 3
dsNotApplicable = 4

DoorAlarmStates enum:
dasUnknown = 0
dasActive = 1 ' i.e. Door IS Forced
dasInactive = 2 ' i.e. Door is NOT Forced

MaskStates enum:
msUnknown = 0
msMasked = 1
msUnmasked = 2
msNotApplicable = 3

AccessCycleStates enum:
acsUnknown = 0
acsInProgress = 1
acsNotInProgress = 2

IPLocksetState

SequenceNumber (int)
MessageType = MessageTypes.IPLocksetChangeOfState
LocksetId (Guid)
CommState (IPLocksetCommStates enum)

Enum IPLocksetCommStates:
  iplcsUnknown = 0
  iplcsOffline = 1
  iplcsOnline = 2
  iplcsException = 3

Alarm

SequenceNumber (int)
MessageType = MessageTypes.Alarm
AlarmId (Guid)
AlarmDate (Datetime)
SourceType (SourceType enum)
SourceId (Guid)
EventType (Event type enum)
Priority (int)
AlarmDescription (string)
AlarmLocation (string)
AlarmDetail (string)
Cardholder = (string)
CardNumber = (long)
FacilityCode = (int)
AlarmMessage = (string)
SoundFile (string)
Acknowledged (boolean)
Cleared (boolean)
Unsecured (boolean)
SiteName (string)
SiteId (Guid)
AcknowledgedByUserName = (string)
AcknowledgedByComputerName (string)
Comments = (string)
RequireComments (bool)

SourceTypes enum: defined above.