Developers Guide

With PINgrid, PINphrase & PINpass Technology

Product Version: 3.3.5816.0

Publication date: April 2019

Authlogics, 12th Floor, Ocean House, The Ring, Bracknell, Berkshire, RG12 1AX, United Kingdom
UK Tel: +44 1344 568 900 US Tel: +1 857 214 2174 email: info@authlogics.com web: http://authlogics.com/

Table of Contents

Introduction

Authlogics Authentication Server is a multi-factor authentication system which provides:

- Token and token-less multi-factor authentication.
- Award winning transaction signing / verification technology.
- Self-service password reset and unlocking.
- Web Service API and RADIUS interfaces for connectivity.
- Authentication technologies:
    - PINgrid Pattern Based Authentication.
    - PINphrase Random Character Authentication
    - PINpass OATH (TOTP) Compliant Authentication

Authlogics Authentication Server has been designed to work with the following directory services:

- Microsoft Active Directory (no schema extensions required)
- Microsoft SQL Server 2016 (Express included)

Guidance

This developers guide provides detailed information about how to leverage the Authlogics Authentication Server Web Services Application Programming Interface (WSAPI). It should be used in conjunction with the “Authlogics Authentication Server Installation and Configuration Guide” which is designed to be an infrastructure document.

Multi-factor Authentication Technology

Background

As the usage of Information Technology has increased exponentially, the need for security of these systems has increased proportionately. Traditionally, authenticating users was solely performed by the user providing a valid username and password. This is known as single-factor authentication as the user “knows” all parts of the authentication process. Passwords have proven to be unsecure and therefore additional authentication factors have become a requirement.

The increase of security provided by multi-factor (typically 2 factor) is that users must now “have something” and “know something” in the authentication process. The “have something” usually takes the form of a physical device, like a key fob, which generates a specific unique One Time Pin (OTP). This OTP must also be entered as part of the authentication process.

Although, these token devices have improved security significantly, they do have certain limitations and incur a costing overhead in both their implementation as well as on-going maintenance. Devices, although in some cases are relatively inexpensive individually, increase the total cost ownership significantly for large deployments as all users may require a device. Furthermore, these devices will eventually fail, require new batteries or get lost or damaged and require replacement. Furthermore, each device must be specifically provisioned to a user, and registering devices against users as well as physically delivering these devices increases the overall expense.

Possibly the biggest issue and security loophole with a key-ring token device solution is that the user must have the device with them continuously. If the device is not within touching distance, unknown to the owner it could be in the hands of an unscrupulous person and often the OTP is clearly visible and not obfuscated in any manner.

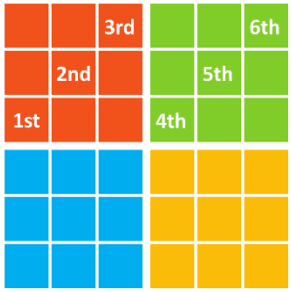
PINgrid Technology

PINgrid technology mitigates the security limitations of the traditional OTP tokens by generating a One Time Code derived from a grid of numbers. These grids are specific to each user and change every minute reflecting different numbers. The additional security of PINgrid is that the user also needs to know a unique pattern to extrapolate an OTP.

To thwart automated brute force attacks, Authlogics includes “Account Lockout” functionality where a user’s account is locked out either indefinitely or for a pre-configured period when a passcode is entered incorrectly after a number of times. PINgrid even mitigates the threat of key-logging, screen scraping and shoulder surfing attacks.

PINgrid is available in 1.5 factor, 2 and 3 factor authentication methodologies. Grids can be views within an app, on a web page, sent via TEXT/SMS or email, or used totally offline via the Authlogics Authenticator in the App Store.

How it works?



Users pattern *Pattern on a challenge grid*

One Time Code is: *133125*

In a 'Prove it!' situation the pattern is used with a challenge grid

- A One Time Password (OTP) is hidden in the grid
- Only the person who knows the secret pattern can 'see' the OTP

Finally, PINgrid technology is truly a One Time Pin authentication solution as all valid passcodes entered can be used only once, even if the authentication attempt occurs within the same time period from the same device.

PINphrase Technology

PINphrase uses a few authentication methods which have become a de facto standard in the banking industry to provide a simple to use but efficient and cost-effective solution.

PINphrase is based upon a passphrase question and answer system which prompts the user to enter random characters from the answer to a randomly chosen question. Unlike passwords, the answers to the questions are typically things that the user is not likely to forget which reduces helpdesk calls, limits resets and further cuts costs. Since the user is only ever entering part of the answer, e.g. letters 2, 5 and second last character. During each login the user is asked to enter different letters, and from a different answers, making the response a One Time Code. The full answer is not revealed during the login, this makes PINphrase ideal for both a 1.5 and 2 factor authentication. PINphrase can also be configured to randomly select letters from different questions to further enhance security.

An administrator can configure multiple common questions for things users will generally know an answer for, and can then specify how many of the questions a user must provide an answer for, e.g. the user must provide answers for at least 4 of the 10 supplied questions. By default, a user is assigned a Codeword which is a randomly chosen dictionary word which can be used for first login.

Scenarios

A new user called Bob Jones is enabled and his mobile phone details are recorded. He then provides answers to at least 6 questions from a pool, he chooses the following:

Place of birth?	Seattle
Pets name?	Tigger
Memorable place?	Springfield
Mother's maiden name?	Watson
Memorable date and time (YYYYMMDDHHMM)	201101021937
First school?	Winchester

Authentication Scenario #1 - 1.5 Factor Authentication

Bob wants to logon to an Internet banking site. He types in his Username and is then presented with a question from the answered pool and is asked to enter specific characters from the answer.

```
Please provide the 1st, 3rd, 4th and the last characters from your
memorable place.
```

To authenticate, Bob will enter `S R I D`.

Authentication Scenario #2 - 2 Factor Authentication

This requires a physical device on which Bob will receive the question and random positions, i.e. the soft token. Typically, this device is a mobile phone as the mobile phone number is unique to the user.

Bob accesses the logon page of his internet banking site and types in his username. Once Bob enters his Username, the PINphrase server detects that the logon process for Bob has started. A challenge will be generated and sent as a SMS/ Text message to Bob's mobile device as follows:

```
PINphrase: Please provide the 2nd, 3rd, 5th and penultimate
characters from your place of birth.
```

To authenticate, Bob will enter `A L S R`.

A key part Authlogics PINphrase is that both the 1.5 and 2 Factor methods have an identical look and feel to the user with the only difference being where the challenge message is displayed.

In cases where mobile phone reception cannot be guaranteed and instant message retrieval may not always be possible, PINphrase can Pre-send tokens. Pre-sending tokens ensures that the user always has a token on his/her device prior to the authentication attempt. As soon as the token is used, then the next token is sent to the user's mobile device ready to be used for the next login.

PINpass Technology

Authlogics PINpass is an OATH RFC compliant 2 factor authentication solution which utilises soft tokens to reduce the costs associated with hardware key fobs. PINpass OTPs are delivered to mobile phones via SMS text messages or as an email for even more flexibility and cost savings.

PINpass gives administrators the ability to pre-send one or more OTP's so that the user always has an OTP on their mobile device prior to logging on. As soon as the last OTP is used, then a new set of OTPs are sent to the user ready for future logon attempts. Alternatively, PINpass can be used totally offline via the Authlogics Authenticator in the App Store.

To increase security and convenience, administrators can configure users to provide an Active Directory password or static PIN with the One Time Pin. A static pin can be entered, before, after or even in the middle of the OTP code making it more difficult for a key logger to differentiate between the OTC code and the users static PIN.

When a user is configured with a real-time token and attempts to login, they enter their unique login name and PINpass sends a 6 to 8 digit OTP to their mobile phone via SMS or email address. The user then enters the OTC along with either they AD password or a static PIN number, depending on the configuration.

The login process is similar for a user who is configured with a pre-send token except that a code is not sent to the user after they enter their user name as they will already have a code on their phone. Instead a new code is only sent after they login for use during the next login.

Web Services Communication

Authlogics Authentication Server supports authentication with a Web API via the following protocols:

- SOAP 1.1
- SOAP 1.2
- HTTP GET
- HTTP POST

By default, the Web Services run on TCP:14000 for HTTP (without encryption) and TCP:14443 for SSL (with encryption). An SSL certificate MUST be installed onto the server and bound to the IIS web site to utilise HTTPS.

Both IPv4 and IPv6 are supported for communication with Web Services.

A single Web API interface is available for common logon processing capabilities a complete API set for managing and automating all server functions. A complete Web Services Description Language (WSDL) listing for the Web Service can be accessed via:

http://{ServerName}:14000/Services/?wsdl

Authentication and Transaction Verification / Signing

The following 2 Web Service operations can be used to process an authentication request and verify a transaction:

- AuthenticateUser
- VerifyTransaction

AuthenticateUser

To process an authentication request via Web Services simply supply the accountname and passcode to the AuthenticateUser function and it will return a status code from the ***Return Codes*** table above.

Example of an HTTP GET authentication validation request…

http://{ServerName}:14000/Services/wsapi.asmx/AuthenticateUser?accountname=adamj&passcode=123456

…which returns the following…

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://{ServerName}:14000/Services/wsapi.asmx/">2</int>
```

…indicating an invalid login.

VerifyTransaction

To verify a transaction via Web Services, supply the accountname, passcode and transaction specific data to the VerifyTransaction function. This operation will return a status code from the ***Return Codes*** table above.

Example of an HTTP GET transaction verification request…

http://{ServerName}:14000/Services/wsapi.asmx/VerifyTransaction?accountname=adamj&passcode=123456&transactionData=1234567890

…which returns the following…

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://{ServerName}:14000/Services/wsapi.asmx/">2</int>
```

…indicating an invalid transaction verification.

Return Codes

0	Access Granted.	Credentials are valid.
1	Access Denied.	Account name not found.
2	Access Denied.	Invalid passcode.
5	Access Denied.	Account Expired.
7	Access Denied.	Account Disabled, Locked out or not valid at this time.
8	Access Denied.	Authentication not available due to a licencing issue.
13	Access Granted.	A pattern change is required.
111	Access Denied.	Directory related error.

PINgrid Challenge Grid Generation

Integrating a PINgrid 1.5 Factor challenge into a web page or application is a simple as calling the GetPinGridToken.ashx Web Service with an HTTP GET request. The GetPinGridToken.ashx Web Service accepts the following parameters which are all optional and can be combined as required:

- accountname
- format
- resolution
- background
- q1, q2, q3, q4

The theme of the generated image is determined by the Global Settings in MMC.

accountname

The “accountname” parameter will generate a challenge specific for the user defined in “accountname”.

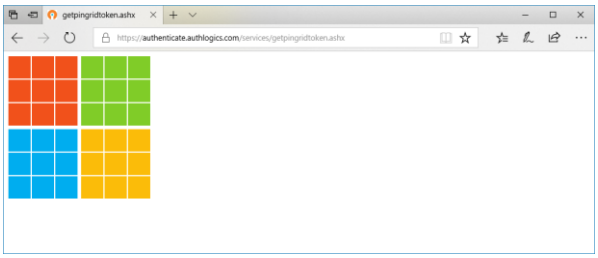
To generate a blank PINgrid grid call the GetPinGridToken.ashx Web Service without the “accountname” parameter, or a blank accountname value, e.g.:

http://{ServerName}:14000/Services/GetPinGridToken.ashx

Note

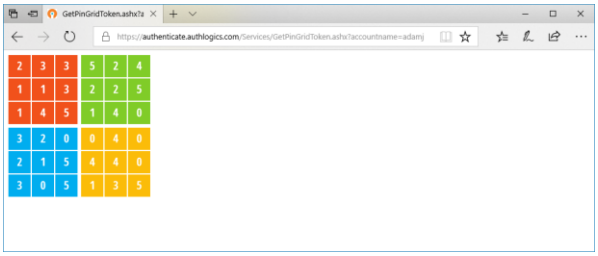
The legacy parameter value “username” may still be used instead of “accountname” but it may be removed in a future version.

The Web Service will return an image as follows which is appropriate to show when the accountname is not specified.



By adding a value for the accountname parameter, numbers are placed on the PINgrid grid or PINphrase challenge text is produced. Even if a user account doesn't match the name provided a challenge will still be generated so not to disclose the existence of an actual user account. e.g.:

http://{ServerName}:14000/Services/GetPinGridToken.ashx?accountname=adamj



Format

The "format", and optional "background", parameters determines the graphical image type of the challenge grid and accepts the following options:

- PNG (default format when parameter is not specified)
- BMP
- JPG
- GIF
- TXT (this returns the values for a grid in plain text and does not return a graphic image)

The image is always a square 1:1 ratio.

http://{ServerName}:14000/Services/GetPinGridToken.ashx?format=gif

Resolution

The "resolution" parameter sets the resolution of the generated PINgrid image. This should match the size of the HTML image object where the image is being displayed to prevent browser rescaling which may result in a fuzzy or blurred image. If the "resolution" parameter is not specified the resolution set in the MMC will be used.

http://{ServerName}:14000/Services/GetPinGridToken.ashx?resolution=150

> **Note**
> If the specified resolution is greater than 2500 then a resolution of 2500 will be used. If the specified resolution is less than 50 then Global Settings resolution will be used.

Background

The optional “background” parameter sets the background HEX colour to use when a non-transparent image type is used, .i.e BMP and JPG. If the “background” parameter is not specified the resolution set in the MMC will be used.

```
http://{ServerName}:14000/Services/GetPinGridToken.ashx?format=jpg&background=FFFFFF
```

Custom HEX colour codes can be found here:

- http://www.colorpicker.com/
- http://www.w3schools.com/tags/ref_colorpicker.asp

Quadrant Colours

The HEX colour of each PINgrid quadrant can be specified via the “q1”, “q2”, “q3” and “q4” parameters. The values specified override the global settings colours set in the MMC.

```
http://{ServerName}:14000/Services/GetPinGridToken.ashx?1=dd4120&q2=31dd20&q3=2090dd&q4=ddc320
```

PINphrase Challenge Generation

Integrating a PINphrase 1.5 factor challenge question into a web page or application is a simple as calling the GetPinPhraseToken.ashx Web Service with an HTTP GET request. The Authlogics Authentication Server will return a challenge string.

The GetPinPhraseToken.ashx Web Service only accepts a “accountname” parameter.

accountname

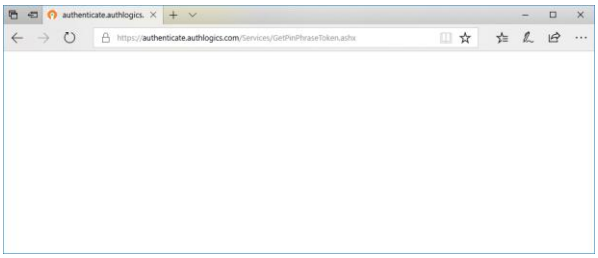
The “accountname” parameter will generate a challenge specific for the user defined in “accountname”.

To generate a blank PINphrase challenge call the GetPinPhraseToken.ashx Web Service without the “accountname” parameter, or a blank accountname value, e.g.:

http://{ServerName}:14000/Services/GetPinPhraseToken.ashx

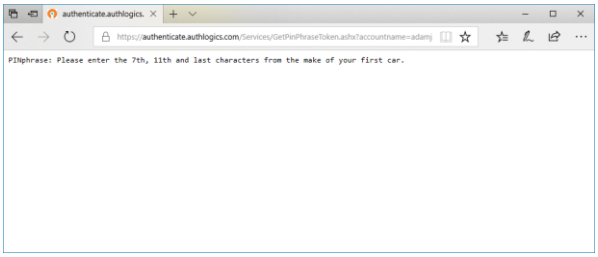
> **Note**
> The legacy parameter value “username” may still be used instead of “accountname” but it may be removed in a future version.

The Web Service will return a blank string as follows which is appropriate to show when the accountname is not specified.



By adding a value for the username parameter, a challenge string is returned. Even if a user account doesn’t match the name provided, a challenge string will still be generated so as not to disclose the existence of an actual user account. e.g.:

http://{ServerName}:14000/Services/GetPinPhraseToken.ashx?accountname=adamj

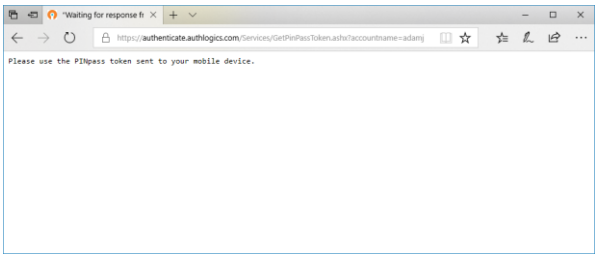


PINpass Challenge Triggering

While PINpass cannot generate a 1.5 factor challenge, the web service call will trigger the sending of a server generated token is appropriate for the user.

The GetPinPhraseToken.ashx Web Service only accepts a “accountname” parameter.

http://{ServerName}:14000/Services/GetPinPassToken.ashx?accountname=adamj



Web Services API (wsapi.asmx)

In addition to the Authlogics Management Console, Authlogics Authentication Server also includes a flexible Web Services Application Programming Interface (WSAPI) which allows for simple user account and server configuration management from remote systems. WSAPI allows for integration with workflow, change management processes and automatic provisioning systems without manual intervention via the Authlogics Management Console.

> **Note**
> Some API functions make use of enums based properties. Enum property values are case sensitive and will fail if the wrong case is used.

Authenticating to the WSAPI

The WSAPI interface requires HTTP Challenge authentication for certain function calls and to access certain property values. Some operations can be performed without authentication whereas others require authentication with an Administrator, an Operator or the actual user.

The web server used by Authlogics is IIS running on Windows. By default the Authlogics IIS web site only allows Windows Authentication for HTTP 401 logons to ensure that passwords cannot be intercepted if SSL is not used.

If the WSAPI client is not able to authenticate using Windows Authentication (Kerberos or NTLM) then either Basic or Digest authentication may be enabled in the IIS web server. A SSL certificate should be installed on the web server and all WSAPI connections should be made using HTTPS to secure the logon credentials to the web server.

WSAPI Function list

The following is a list of all available WSAPI functions:

1. AuthenticateRadiusUser **(Internal use only)**
2. AuthenticateUser
3. AuthenticateUserAdPasswordless **(Internal use only)**
4. CheckPasswordAgainstPolicy
5. CreateRealm
6. CreateUser
7. CreateUserEx
8. DeleteRealm
9. DeleteUser
10. DeliverMessage
11. DisableEmergencyOverride
12. DisablePinGrid
13. DisablePinPass
14. DisablePinPhrase
15. EnableEmergencyOverride
16. EnablePinGrid
17. EnablePinPass
18. EnablePinPhrase
19. GenerateNewUserSeed
20. GetRealms
21. GetSettingsProperty
22. GetUserProperty
23. GetUserStatus **(Internal use only)**
24. PinGridChangeMIP
25. PinGridGenerateMIP
26. PinGridProvision
27. PinPassGeneratePIN
28. PinPassProvision
29. PinPhraseAddQuestion
30. PinPhraseEditQuestion
31. PinPhraseGenerateCodeword
32. PinPhraseProvision
33. PinPhraseRemoveQuestion
34. RealmExists
35. RenameRealm
36. RenameUser
37. SendHTMLPinGridLetter
38. SendHTMLPinPassLetter
39. SendHTMLPinPhraseLetter
40. SendPresendToken
41. SendRealTimeToken
42. SendRealTimeTokenbyProduct

43. SendToken
44. SetADpassword
45. SetSettingsProperty
46. SetOnlineVaultPassword **(Internal use only)**
47. SetUserProperty
48. TokenHardwareAdd
49. TokenHardwareEnabled
50. TokenHardwareRemove
51. VerifyTransaction
52. WebPortalVerifyOTP **(Internal use only)**

WSAPI Function Details

Function: `AuthenticateUser`

The AuthenticateUser function processes a logon request for a user account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp andyp@sample.com	A user account name which is resolvable in the directory.
passcode	string	123456	A passcode to authenticate the account.

An integer code is returned indicated the result of the authenticate request. These codes are based upon industry standard RADIUS return values.

0	Access Granted.	Credentials are valid.
1	Access Denied.	Account name not found.
2	Access Denied.	Invalid passcode.
5	Access Denied.	Account Expired.
7	Access Denied.	Account Disabled, Locked out or not valid at this time.
8	Access Denied.	Authentication not available due to a licencing issue.
13	Access Granted.	A pattern change is required.
111	Access Denied.	Directory related error.

Function: `CheckPasswordAgainstPolicy`

The CheckPasswordAgainstPolicy function checks a supplied password against the configured Password Policy on the Authlogics Server; it does NOT attempt to set the supplied password.

To configure the policy apply a Group Policy containing the Password Policy Agent template to the authentication server computer account. This is typically the same policy which is applied to the Domain Controllers.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name in the directory.
password	string	Pa55w0rd	A sample password to test.

Function: `CreateRealm`

The CreateRealm function creates an Authlogics Realm using name provided.

Parameter	Type	Value Sample	Description
newRealm	string	myrealm myrealm.com	A realm name to contain user accounts.

Function: `CreateUser`

The CreateUser function creates an Enabled Authlogics user account using default values. It does not configure the account for any authentication types.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name to store in the directory.

Function: `CreateUserEx`

The CreateUserEx function is an expanded version of the CreateUser function which creates an Enabled Authlogics user account using default values and allows updating common properties during the creation. It does not configure the account for any authentication types.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name to store in the directory.
firstName	string	Andy	User First name.
lastName	string	Pearson	User Last name.
mailAddress	string	andyp@sample.com	A valid email address for the user.

Function: `DeleteRealm`

The DeleteRealm function removes an Authlogics Realm using name provided.

Parameter	Type	Value Sample	Description
oldRealm	string	myrealm myrealm.com	A realm name to be removed. Note: The realm cannot be removed if it contains user accounts.

Function: `DeleteUser`

The DeleteUser function deletes an Authlogics user account from the directory including all its settings and attributes. When using Active Directory it does NOT delete the actual AD user account, only the Authlogics metadata is removed off of the account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `DeliverMessage`

The DeliverMessage function delivers a message to the user via SMS or email.

Parameter	Type	Value Sample	Description
deliveryMethod	Enum	SMS Email	The required delivery method.
toAddress	string	user@domain.com +15551234	An email address or phone number for the recipient.
emailSubject	string	A Subject Line	The subject of an email message. Note: This is ignored when sending an SMS.
message	string	A bunch of text	The message body to send to the user. Note: A long message may trigger multiple SMS messages depending on the SMS provider used.

Function: `DisableEmergencyOverride`

The DisableEmergencyOverride function disables the Emergency Override setting on a user account. If Emergency Override is not enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `DisablePinGrid`

The DisablePinGrid function disables the use of PINgrid on a user account. If PINgrid is not enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `DisablePinPass`

The DisablePinPass function disables the use of PINpass on a user account. If PINpass is not enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `DisablePinPhrase`

The DisablePinPhrase function disables the use of PINphrase on a user account. If PINphrase is not enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `EnableEmergencyOverride`

The EnableEmergencyOverride function enables the Emergency Override functionality on a user account. If *UseADpassword* is set to *True* then the *EmergencyOverrideCode* value is ignored.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
EmergencyOverride ExpiryMethod	Enum	nLogins	The method used to expire the use of Emergency Override. This can be a combination of number of logins (*nLogins*), a period of time (*TimePeriod*) or both (*nLoginsorTimePeriod*).
UseADpassword	Boolean	False	Set the account to use the AD password instead of a set code/password. Note: This feature is only available in Active Directory environments.
EmergencyOverride Code	string	5ecr3t	The code/password to be used for Emergency Override.

Function: `EnablePinGrid`

The EnablePinGrid function enables the use of PINgrid on a user account. If PINgrid is enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `EnablePinPass`

The EnablePinPass function enables the use of PINpass on a user account. If PINpass is enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `EnablePinPhrase`

The EnablePinPhrase function enables the use of PINphrase on a user account. If PINphrase is enabled on the account then this function has no effect.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `GenerateNewUserSeed`

The GenerateNewUserSeed function generates a new 256bit seed on a user account. If a seed already exists then it will be replaced with the new one.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `GetRealms`

The GetRealms function retrieves a string array of realms that exist in the directory. This is a read only function.

Parameter	Type	Value Sample	Description
n/a			

Function: `GetSettingsProperty`

The GetSettingsProperty function retrieves the value/values of various global settings properties. This is a read only function, to set the value of a property use the SetSettingsProperty function.

Multiple values can be queried at once by specifying all the required properties in a CSV string.

Parameter	Type	Value Sample	Description
names	string	SMTPServer1, SMTPPort1	The property names to access. Note: Leaving this value blank returns a list of supported property values.

Valid values for the names parameter which do not require authentication (Anonymous):

```
SchemaVersion,ToleranceLevel, TolerancePeriod, LockoutDuration, LockoutThreshold,
LockoutReset, AllowResetMasterPassword, UnlockMasterAccountOnPasswordReset,
AllowUpdateMobilePhoneNumber, AllowTokenDeviceChange, ADUsernameCustomAttribute,
GUIDAdministrators, GUIDOperators, GUIDAdministrators, GUIDServers, GUIDRadius,
GUIDADPassthrough, SMTPServer1, SMTPServer2, SMTPPort1, SMTPPort2, SMTPFromAddress,
SMTPEnableSSL, SMTPUseWindowsCredentials, SMSEnabled, SMSSendLimit,
SMSDefaultCountryCode, SSPURL, RealTimeTokenLifespan, AllowEmergencyOverride,
MaxOverrideTime, MaxOverrideUses, PasswordVaultEnabled, DirectoryID,
PinGridMatrixMinNumberOfSquares, PinGridMatrixTheme, PinGridMIPHistory,
PinGridMIPMaxAge, PinGridMIPMinLength, PinGridMIPMinAge, PinGridMIPComplexity,
PinGridMIPMaxAdjacencies, PinGridMIPMaxCellRepeats, PinGridMIPMinNumberOfQuadrants,
PinGridHASHLevel, PinGridMessagePrefix, PinGridMatrixFontSize,
PinGridMatrixColourQ1, PinGridMatrixColourQ2, PinGridMatrixColourQ3,
PinGridMatrixColourQ4, PinGridMatrixBitmapSizeDPI, PinGridMatrixHTMLEmail,
PinPhraseMinNumberOfQuestions, PinPhraseMinAnswerLength, PinPhraseQuestions,
PinPhraseMessagePrefix, PinPhraseUseMultipleQuestionsPerLogin,
PinPassMessagePrefix, PinPassMinLength, PinPassPINMinLength, PinPassPINPosition,
PinPassPINEnforced, RADIUSFilterEnabled, ADPassthroughEnabled.
```

Valid values for the names parameter which require Admin rights:

```
SMTPUsername
```

Function: `GetUserProperty`

The GetUserProperty function retrieves the value/values of various user account properties. This is a read only function, to set the value of a property use the SetUserProperty function.

Multiple values can be queried at once by specifying all the required properties in a CSV string.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
names	string	FirstName,LastName	The property names to access. Note: Leaving this value blank returns a list of supported property values.

Valid values for the names parameter which do not require authentication (Anonymous):

```
UPN, FirstName, LastName, Realm, Exists, Enabled, APL, ValidFrom, ValidTo,
Description, PinGridEnabled, PinGridProvisioned, PinGridMIPMustChange,
PinGridMIPNeverExpires, PinGridRequireRemoteSeed, PinGridRequire2FA,
PinGridEnable2FA, PinGridDelivery, PinGridQueueType, PinPhraseEnabled,
PinPhraseProvisioned, PinPhraseAnswersMustChange, PinPhraseRequire2FA,
PinPhraseEnable2FA, PinPhraseDelivery, PinPhraseQueueType, PinPassEnabled,
PinPassProvisioned, PinPassPINMustChange, PinPassRequireRemoteSeed,
PinPassDelivery, PinPassQueueType, PinPassTokensPerMessage.
```

Valid values for the names parameter which require Admin or Operator rights:

```
LockedOut, EmergencyOverrideEnabled, PinGridMIPCreationDate, PinGridMIPExpiryDate,
PinGridMIPdaysSinceLastChanged, PinGridTokenLifespan, PinPhraseCodeLength,
PinPhraseTokenLifespan, PinPassTokenLifespan.
```

Valid values for the names parameter which require Admin or Operator rights, or can be accessed by the actual user:

```
MobileNumber, MobilePrivate, MailAddress, RemoteSeed, PinGridMatrixNumberOfSquares,
PinPhraseAnswers, PinPassPIN, PinPassCodeLength, PinPassPINisADpassword.
```

Function: `PinGridChangeMIP`

The PinGridChangeMIP function sets a new PINgrid pattern on the user account. The pattern is specified in MIP comma separated notation of grid positions and is stored as a hash and once written, **CANNOT be retrieved in plain text**.

Note: This function may be called by any authenticated user so long as they are attempting to update their own MIP, otherwise Authlogics Admin rights are required.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
gridSize	string	6	The X or Y size of the grid use for the new pattern. *6* or *8* are the only accepted values.
MIP	string	23,29,35,24,30,36	A comma separated list of numbers denoting the positions in a grid for the pattern.

Function: `PinGridGenerateMIP`

The PinGridGenerateMIP function creates a new random pattern in MIP comma separated notation. Either a simple or a complex pattern can be generated.

The new MIP is returned by the function as a string and is not written to a user account. The new MIP can be used on a user account within the *PinGridChangeMIP*, *PinGridProvision* and *SendHTMLPinGridLetter* functions.

Parameter	Type	Value Sample	Description
gridSize	integer	6	The X or Y size of the grid use for the new pattern. *6* or *8* are the only accepted values.
complexPattern	boolean	False	To generate a complex pattern set this value to *True*. For a simple pattern set it to *False*.

Function: `PinGridProvision`

The PinGridProvision function provisions a user account for user with PINgrid. This must be done at least once to allow PINgrid to be used with the account. The inputs are very similar to the PinGridChangeMIP function however this function does more than just setting the MIP.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
gridSize	integer	6	The X or Y size of the grid use for the new pattern. *6* or *8* are the only accepted values.
MIP	string	23,29,35,24,30,36	A comma separated list of numbers denoting the positions in a grid for the pattern.
OverrideRestrictions	boolean	False	Override the pattern complexity restriction checks when setting the pattern. It is not recommended to override the built in complexity settings as this could lead to lower security.

Function: `PinPassGeneratePIN`

The PinPassGeneratePIN function generates a random PIN which complied to the PINpass policy.

Parameter	Type	Value Sample	Description
n/a			

Function: `PinPassProvision`

The PinPassProvision function provisions a user account for user with PINpass. This must be done at least once to allow PINpass to be used with the account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
PIN	string	7651	A PIN (or password) to be used as the knowledge component for authentication. If no PIN is specified then a random PIN will generated and saved to the account.
PINisADpassword	boolean	False	Use the Active Directory password instead of a PIN. If this value is set to True then the PIN value, if specified, will not be used. Note: This option is only available in an Active Directory Environment.
OTPcodeLength	integer	6	The length of the OTP code which will be sent to the user. As per OATH RFC specifications, accepted values are 6, 7 and 8 only.

Function: `PinPhraseAddQuestion`

The PinPhraseAddQuestion function adds a new question to the list available for users to provide answers to.

Parameter	Type	Value Sample	Description
question	string	Your favourite colour	A string containing the actual question to add.

Function: `PinPhraseEditQuestion`

The PinPhraseEditQuestion function alters an existing question in the list available for users to provide answers to.

Warning: This should only be used for minor changes and as existing answers will still be matched to the question. For brand new questions you should delete the old one and add a new one.

Parameter	Type	Value Sample	Description
oldQuestion	string	Your best colour	A string containing the old actual question.
newQuestion	string	Your favourite colour	A string containing the new actual question.

Function: `PinPhraseGenerateCodeword`

The PinPhraseGenerateCodeword function returns a random word from the PINphrase dictionary file.

Parameter	Type	Value Sample	Description
n/a			

Function: `PinPhraseProvision`

The PinPhraseProvision function provisions a user account for user with PINphrase. This must be done at least once to allow PINphrase to be used with the account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
codeWord	string	England	A string to be used as the answer to the first question, which the default question is “your code word”. If no answer is specified then a random word will be selected from a dictionary file and saved to the account as the answer to the first question.
OTPcodeLength	integer	3	The number of letters from the answer which will be requested from the user to create their OTP code. Recommended values are between 3 and 5.

Function: `PinPhraseRemoveQuestion`

The PinPhraseRemoveQuestion function removes an existing question from the list available for users to provide answers to. Any existing answers to the removed question will be removed from the user account only when the account is next updated. A bulk answer delete will NOT be triggered although the answer will be ignored during subsequent processing.

Parameter	Type	Value Sample	Description
question	string	Your favourite colour	A string containing the actual question.

Function: `RealmExists`

The RealmExists function checks if the specified realm exists in the directory.

The function returns a *True* or *False* Boolean value.

Parameter	Type	Value Sample	Description
realm	string	myrealm.com	A realm name which may or may not exist.

Function: `RenameRealm`

The RenameRealm function renames an Authlogics realm in the directory.

Note: RenameRealm cannot be used in an Active Directory environment.

Parameter	Type	Value Sample	Description
oldRealm	string	oldrealm.com	A realm name which is resolvable in the directory.
newRealm	string	newrealm.com	The new realm name to be renamed to.

Function: `RenameUser`

The RenameUser function renames an Authlogics user account in the directory.

Note: RenameUser cannot be used in an Active Directory environment. The account must be renamed directly in AD.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
newName	string	AndyK	The new account name to be renamed to.

Function: `SendHTMLPinGridLetter`

Sends an HTML formatted “PINgrid welcome letter” to the user via email from the Authlogics server providing details of how to use the account. The email is sent to the email address specified on the user account only.

The MIP must be specified here as it cannot be retrieved from the user account in plain text. This function should be called directly after calling the PinGridChangeMIP or PinGridProvision functions while the MIP is still in memory as plain text.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
templateName	string	PINgridUserTemplate	The name of the HTML template file to be used as a base for the email. The file name can be specified with or without the HTML file extension. The file must be available in the Authlogics Program Files folder.
MIP	string	23,29,35,24,30,36	A comma separated list of numbers denoting the positions in a grid for the pattern.

Function: `SendHTMLPinPassLetter`

Sends an HTML formatted “PINpass welcome letter” to the user via email from the Authlogics server providing details of how to use the account. The email is sent to the email address specified on the user account only.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
templateName	string	PINpassUserTemplate	The name of the HTML template file to be used as a base for the email. The file name can be specified with or without the HTML file extension. The file must be available in the Authlogics Program Files folder.

Function: `SendHTMLPinPhraseLetter`

Sends an HTML formatted "PINphrase welcome letter" to the user via email from the Authlogics server providing details of how to use the account. The email is sent to the email address specified on the user account only.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
templateName	string	PINphraseUserTemplate	The name of the HTML template file to be used as a base for the email. The file name can be specified with or without the HTML file extension. The file must be available in the Authlogics Program Files folder.

Function: `SendRealtimeToken`

Sends a server generated real time token to the user based on the configured authentication provider only if they are configured for Real-Time token use. The token may be sent via email or Text Messaging depending on the user settings.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `SendRealtimeTokenbyProduct`

Sends a server generated real time token to the user only if they are configured for Real-Time token use. The token may be sent via email or Text Messaging depending on the user settings.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
product	enum	PinGrid	Specify the type of authentication technology to send a token for.

Valid values for the product parameter include:

`PinGrid, PinPhrase, PinPass`

Function: `SendToken`

Sends a server generated token to the user based on the configured authentication provider and queue type (Real-Time / Pre-Send). The token may be sent via email or Text Messaging depending on the user settings.

This function is ideally called when a user is configured to use a pre-send token and the initial tokens need to be delivered.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.

Function: `SetADpassword`

The SetADpassword function updates the Active Directory Domain account password for a user account. This function can be called by the actual user or an Authlogics Administrator.

Note: This function is only available when using AD as the directory.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
ADpassword	string	Pa55w0rd	The new AD password to be written to the Windows Domain.

Function: `SetSettingsProperty`

The SetSettingsProperty function commits the value / values of various global settings properties. This is a write only function, to get the value of a property use the GetSettingsProperty function.

Multiple values can be set at once by specifying all the required properties and values in the corresponding a CSV strings.

Parameter	Type	Value Sample	Description
names	string	SMTPServer1, SMTPPort1	The property names to access. Note: Leaving this value blank returns a list of supported property values.
values	string	mail.server.com, 25	The values to write to the properties specified in *names*.

Valid values for the names parameter which require Admin rights:

```
ToleranceLevel, TolerancePeriod, LockoutDuration, LockoutThreshold, LockoutReset,
AllowResetMasterPassword, UnlockMasterAccountOnPasswordReset,
AllowUpdateMobilePhoneNumber, AllowTokenDeviceChange, ADUsernameCustomAttribute,
GUIDAdministrators, GUIDOperators, GUIDAdministrators, GUIDServers, GUIDRadius,
GUIDADPassthrough, SMTPServer1, SMTPServer2, SMTPPort1, SMTPPort2, SMTPFromAddress,
SMTPEnableSSL, SMTPUseWindowsCredentials, SMTPUsername, SMTPPassword, SMSSendLimit,
SMSDefaultCountryCode, SSPURL, RealTimeTokenLifespan, AllowEmergencyOverride,
MaxOverrideTime, MaxOverrideUses, MaxOverrideUses, PinGridMatrixMinNumberOfSquares,
PinGridMatrixTheme, PinGridMIPHistory, PinGridMIPMaxAge, PinGridMIPMinLength,
PinGridMIPMinAge, PinGridMIPComplexity, PinGridMIPMaxAdjacencies,
PinGridMIPMaxCellRepeats, PinGridMIPMinNumberOfQuadrants, PinGridHASHLevel,
PinGridMessagePrefix, PinGridMatrixFontSize, PinGridMatrixColourQ1,
PinGridMatrixColourQ2, PinGridMatrixColourQ3, PinGridMatrixColourQ4,
PinGridMatrixBitmapSizeDPI, PinGridMatrixHTMLEmail, PinPhraseMinNumberOfQuestions,
PinPhraseMinAnswerLength, PinPhraseMessagePrefix,
PinPhraseUseMultipleQuestionsPerLogin, PinPassMessagePrefix, PinPassMinLength,
PinPassPINMinLength, PinPassPINPosition, PinPassPINEnforced, RADIUSFilterEnabled,
ADPassthroughEnabled.
```

Function: `SetUserProperty`

The SetUserProperty function commits the value / values of various user account properties. This is a write only function, to get the value of a property use the GetUserProperty function.

Multiple values can be set at once by specifying all the required properties and values in the corresponding a CSV strings.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
names	string	FirstName,LastName	The property names to access. Note: Leaving this value blank returns a list of supported property values.
values	string	John,Smith	The values to write to the properties specified in *names*.

Valid values for the names parameter which require Admin or Operator rights:

```
LockedOut, Enabled, MailAddress, ValidFrom, ValidTo, PinGridMIPMustChange,
PinGridMIPNeverExpires, PinGridRequireRemoteSeed, PinGridRequire2FA,
PinGridEnable2FA, PinGridTokenLifespan, PinGridDelivery, PinGridQueueType,
PinPhraseAnswersMustChange, PinPhraseCodeLength, PinPhraseRequire2FA,
PinPhraseEnable2FA, PinPhraseTokenLifespan, PinPhraseDelivery, PinPhraseQueueType,
PinPassPINMustChange, PinPassCodeLength, PinPassRequireRemoteSeed,
PinPassPINisADpassword, PinPassTokenLifespan, PinPassDelivery, PinPassQueueType,
PinPassTokensPerMessage.
```

Valid values for the names parameter which require Admin or Operator rights, or can be accessed by the actual user:

```
MobileNumber, MobilePrivate, PinGridMIP, PinPassPIN, PinPhraseAnswers.
```

Valid values for the names parameter which require Admin rights:

```
FirstName, LastName, Description.
```

Note

LockedOut can only be set to **False** to unlock an account. *LockedOut* cannot be set to **True** manually.

Function: `TokenHardwareAdd`

The TokenHardwareAdd function adds hardware token details to a user account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
enabled	boolean	True	Set the enabled status of the new token. This would typically be set to True for a new device.
tokenType	enum	WindowsPhone	The plaform the soft token is installed on.
tokenID	string	2419216713157481	The hardware / device ID of as indicated by the soft token application.

Valid values for the tokenType parameter include:

`WindowsDesktop, WindowsStore, WindowsPhone, Android, AppleiOS, AppleMacOS, BlackBerry, NokiaSymbian.`

Function: `TokenHardwareEnabled`

The TokenHardwareEnabled function allows for individual tokens to be enabled or disabled on a user account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
enabled	boolean	True	The new enabled status of the token.
tokenID	string	2419216713157481	The hardware / device ID of as indicated by the soft token application.

Function: `TokenHardwareRemove`

The TokenHardwareRemove function removes individual tokens from a user account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp	A user account name which is resolvable in the directory.
tokenID	string	2419216713157481	The hardware / device ID of as indicated by the soft token application.

Function: `VerifyTransaction`

The VerifyTransaction function processes a 3 factor logon request for a user account.

Parameter	Type	Value Sample	Description
accountName	string	AndyP domain\andyp andyp@sample.com	A user account name which is resolvable in the directory.
passcode	string	123456	A passcode to authenticate the account.
transactionData	string	Abcd1234	Transaction string being used as part of the signing process.

Much like the AuthenticateUser function, an integer code is returned indicated the result of the verify transaction request. These codes are based upon industry standard RADIUS return values.

0	Access Granted.	Credentials are valid.
1	Access Denied.	Account name not found.
2	Access Denied.	Invalid passcode.
5	Access Denied.	Account Expired.
7	Access Denied.	Account Disabled, Locked out or not valid at this time.
8	Access Denied.	Authentication not available due to a licencing issue.
13	Access Granted.	A pattern change is required.
111	Access Denied.	Directory related error.

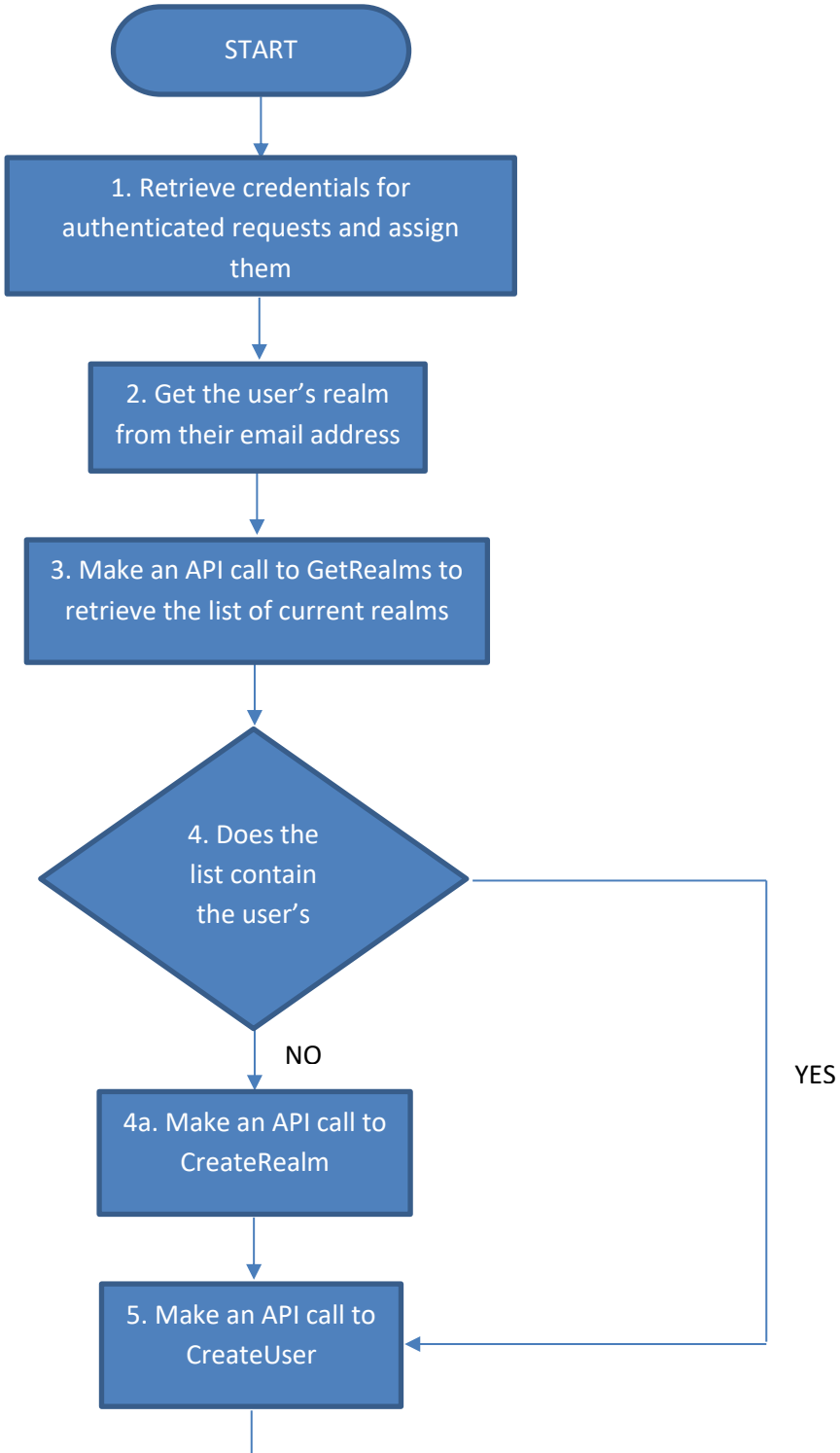
Example: Programmatically creating a user account

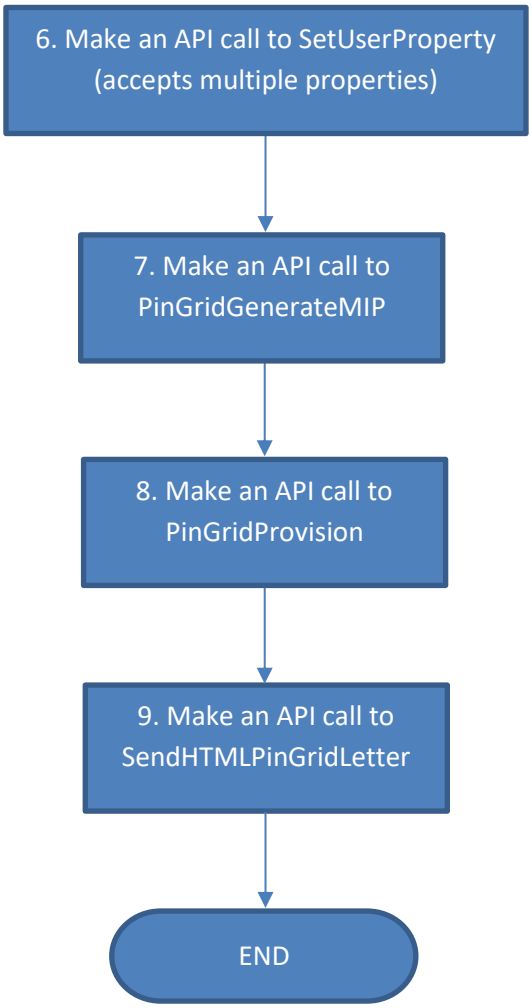
The following example uses a users email address as the basis for creating a new account. It uses the email address suffix as a realm name to create the account in, and if that realm doesn't exist it creates the realm first.

Next a random PINgrid pattern (MIP) will be generated and use to provision the user for PINgrid. Lastly the user will be sent a welcome email containing their new pattern.

> **Note**
> Creating a realm is not possible when using Active Directory as a database.

Process Flow





Explanation

> **Note**
> all API Requests expect their relevant parameters to be passed in the request for them to function correctly – please refer to the API reference for this if necessary.

1. Retrieve the credentials using the method described before, and then assign them accordingly. If using a service reference then you can assign this to this object instance credentials, and it will apply per request. If using a standard web request, then it is on a per request basis, setting them with the *NetworkCredential* object as previously described.
2. There are many ways to retrieve the *realm* from an email address, so it is down to personal preference. E.g. splitting the string on the '@' symbol, or declaring it as a mail address and using the 'host' attribute, or applying a regular expression, etc.
3. The *GetRealms* function requires authentication, so this call needs the credentials applied to it. It can be pulled back into most enumerable objects, an array, a list, or assign it to a variable and let it be implicit.
4. It is sensible to run this check before creating a realm, because if the realm exists then it is a wasted request as the server will merely pass back a message stating that it exists already.

    a. If the realm does not exist, then it needs to be created, so that the user is allocated to the correct database as per their realm.
5. If it does exist, then we can skip creating the realm and go straight to creating the user account.
   Make a request to *CreateUser* to create the user account.
6. Set any additional user properties as needed by calling the *SetUserProperty* function. This function accepts a comma separated list of properties, and a comma separates list of values, so many attributes can be applied to a user at any one time.
7. When calling the *PinGridGenerateMIP* function it is a good idea to store the result in a variable. This can then be utilised further in the next API call to provision the user. This can be made as complicated as you need it to be, and this will depend on your requirements for pattern complexity.
8. Call the *PinGridProvision* function to provision the user for PINgrid. Use the stored value from step 7 for the MIP. When adding parameters for the provision API call, the default value for a grid size is 6 – as in a 6x6 grid.
9. Call the *SendHTMLPinGridLetter* to send the welcome email to the user. Use the stored value from step 7 for the MIP. When adding parameters for the sending of the HTML letter to the user, the *templateName* string value is the default value of the PINgrid template, namely – 'PINgridUserTemplate', if this was a PINpass letter being sent then it would be 'PINpassUserTemplate', and if the letter was PINphrase then the value would be 'PINphraseUserTemplate'.

Using the Web Services API with Visual Studio

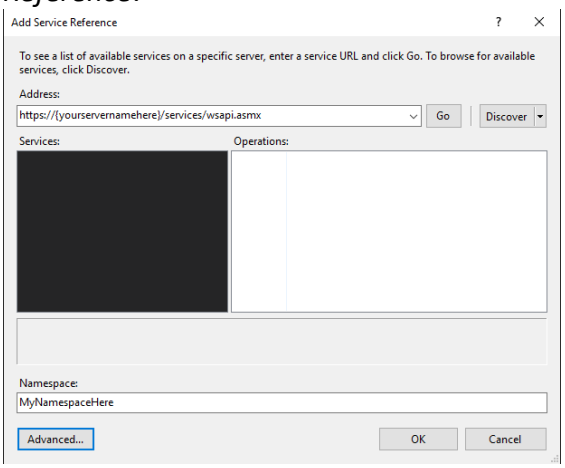
In addition to HTTP Get, Post and SOAP you are also able to communicate with the API via code based references, as well as invoking parts of it from a URL. To do this, we can use a Service Reference or a straightforward web request.

Configuring a Service Reference

For a Service Reference, we are assuming the use of Visual Studio 2012 or higher. **The following examples are written with C# using .NET 4.6.2 – however lower versions can be used, although the code may vary.**

To create a service reference:

(1) Right click on the references section of your project and select *Add Service Reference*.



(2) Fill in the address of the server, as well as the namespace that you wish to work with. Click *OK*.

   a. The advanced options should be left as their default values.

Configuring the web.config for a Service Reference

Adding a service reference (as above) will place additional values into the web.config of your project automatically, such as the binding and the endpoint. There may be extra information, such as a custom binding (not required for this example).

```
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="YourBindingName">
        <security mode="Transport">
          <transport clientCredentialType="Basic" />
        </security>
      </binding>
    </basicHttpBinding>
  </bindings>
  <client>
    <endpoint address="https://yourservername/services/wsapi.asmx"
      binding="basicHttpBinding" bindingConfiguration="YourBindingName"
      contract="YourNamespace.YourBindingName" name="YourBindingName" />
  </client>
</system.serviceModel>
```

Within the *system.serviceModel* section, you will see all of the information required. Any of the extra information should be removed at this point, unless you need to add extra values that are specific to your configuration.

Fill in a suitable *binding name* (defaults such as WSAPISoap may be added, as an example), and make sure the specified value is consistent thought the configuration. Check the *endpoint address* to ensure it is as originally specified when creating the Service Reference.

The most important section of this configuration is *security mode*. This example uses the mode of *Transport*; i.e. that information is requested/sent over HTTPS. The credential type of *Basic* is being used for sending the username and password so that WSAPI methods that require authentication can authenticate.

Note

To use Basic authentication to the WSAPI you must enable Basic Authentication in IIS on the *Services* sub site as only Windows Authentication is enabled by default.

Making an anonymous request with a Service Reference

When making an anonymous/unauthenticated request, we instantiate the webservice so that we may use its methods like a normal class:

```
authenticate.WSAPISoapClient auth = new authenticate.WSAPISoapClient();
```

In this example, the namespace is *authenticate*, and we are using the WSAPISoapClient. We will also be looking at how to use the login validation method.

Next, we will use an asynchronous task to validate our login credentials. These can be passed from a front end screen such as an html form or similar.

```
var logon = await Task.Run(() => auth.AuthenticateUserAsync(context.UserName,
context.Password));
```

To run this asynchronous request, the method it is enclosed in should be marked as asynchronous with the keyword ‘async’ before the method name.
E.g.

```
public async Task
```

This allows us to use the ‘await’ operator, which essentially means we will wait for the task to complete.

The *Task.Run* section of this is to run the AuthenticateUserAsync task that has been generated by the Service Reference when we added it earlier. The lambda expression it is encased in creates a delegate using LINQ – just so we can use it here without having to use another function to return a result.

NB - *Task.Run* is a part of .NET 4.5 onwards, and will need to include *System.Threading*

The result is then returned to the variable *logon*, with which we can then check against to confirm a success or failed logon, and then either log the user in, or give them an error message.

e.g.

```
if (logon != 0)
{
       //This means that the logon has failed - handle it here
       return;
}
else
{
       //0 is a successful logon - continue as you wish to
}
```

For more information on using async operators and how they work visit:
https://msdn.microsoft.com/en-gb/library/mt674882.aspx

For information on using this code before .NET 4.5 visit:
https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.100).aspx

Making an authenticated request with a Service Reference

This is very similar to the above. The extra code needed is the credentials that will be required to authenticate against the method requiring authentication.

Where we instantiate the Service Reference, we can add in the credentials to the request:

```
auth.ClientCredentials.UserName.UserName = "YourUserName";
auth.ClientCredentials.UserName.Password = @"YourPassWord";
```

The credentials do not necessarily have to be specified in plain text within the code. If you are using keys within your web.config file, then they can be applied as well. These should be defined within the appSettings section.

E.g.

```
    <add key="ApiUserName" value="YourUserName" />
    <add key="ApiPassWord" value="YourPassWord" />
```

These can then be utilised in code via the following means:

```
private string apiUserName = WebConfigurationManager.AppSettings["ApiUserName"];
private string apiPassWord = WebConfigurationManager.AppSettings["ApiPassword"];
auth.ClientCredentials.UserName.UserName = apiUserName;
```

So now we have the credentials, and they are assigned to the request. The request is then called in either the manner above, or another example such as the one below:

```
var realmsList = Task.Run(async () => await auth.GetRealmsAsync());

if (realmsList.Result != null)
{
	if (!realmsList.Result.Contains("mytestrealm.com"))
	{
		//Here we can check the result, and see if it contains a value
	}
}
```

The slight difference in this version, is that we are using an async LAMBDA expression, rather than asynchronously calling the task straight out.

This is because the parent method is not, or cannot be marked as async – Therefore we can mark the delegate as asynchronous, and then await the GetRealmsAsync task within that.

The result can then be checked by looking at the result attribute of the variable we created. If it is not null, then it has executed and returned something. We can then do more with that afterwards, such as calling another task.

In the above example, you might call the task to create the realm if it does not already exist.

Advanced Configuration

Advanced configuration options for Authlogics are controlled via the Windows registry and/or the IIS web.config file. The following entries are created during the installation of Authlogics server components and typically most of them should only be changed if instructed by a Authlogics support engineer.

> **Note**
> After changing a registry key on the Authlogics Server the IIS components must be restarted by running `IISRESET` from an elevated admin command prompt.

Changing the server name and port

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\AuthlogicsPort
```

Default Value: `14000 (14443 when SSL is enabled)`

Used by components: Authlogics Server; Management Console; UAG Plug-in

Notes: May be changed but must match the port used by the IIS Web Services.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\AuthlogicsServer
```

Default Value: `localhost` / value provided during setup

Used by components: Authlogics Server; Management Console; UAG Plug-in

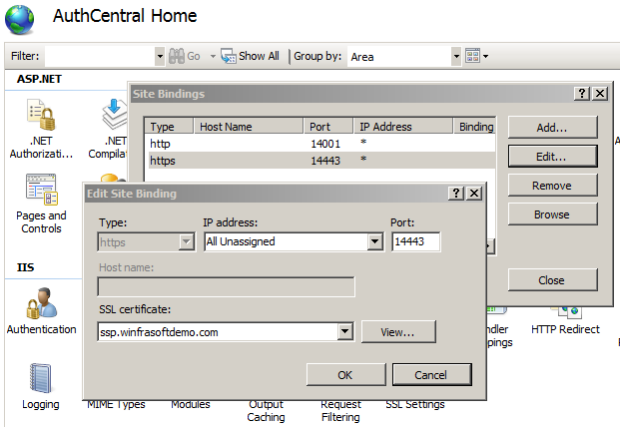
Notes: May be changed but must match the DNS name used by the IIS Web Services.

Configuring SSL for secure connections

To configure Authlogics to use SSL encrypted connections instead of HTTP the following 6 steps must be performed:

(1) Install a SSL certificate on the Authlogics server
   a. The Common Name (CN or SAN) in the certificate must match the DNS value for the *AuthlogicsServer* registry entry or make use of a wide card certificate.
   b. The certificate must be trusted by all systems that connect directly to the Authlogics server.

(2) Edit the HTTPS IIS bindings for the *Authlogics* web site and select the SSL certificate.



a. The HTTPS binding will exist on port 14443 by default.

(3) Remove the HTTP binding to prevent unencrypted communiation (optional).

(4) Change the following registry keys on the Authlogics server, and on any systems where the Authlogics management console is installed:

Registry Key	Value
`HKLM\SOFTWARE\Authlogics\Authlogics Authlogics\AuthlogicsServerName`	DNS name of local server to match Common Name (CN or SAN) in the SSL certificate. The DNS name must resolve to the IP address where IIS is running. E.g.: `my.server.com`
`HKLM\SOFTWARE\Authlogics\Authlogics Authlogics\AuthlogicsPort`	`14443`
`HKLM\SOFTWARE\Authlogics\Authlogics Authlogics\EnableSSLwithAuthlogicsServer`	`1`

(5) Change the following web.config entries on the Authlogics server:

```
C:\Program Files\Authlogics Authentication Server\wwwroot\Web.config
```

Web.config Key	Value
`key="AuthlogicsServerName"`	DNS name of local server to match Common Name (CN or SAN) in the SSL certificate. The DNS name must resolve to the IP address where IIS is running. E.g.: `value="my.server.com"`
`key="AuthlogicsServerPort"`	`value="14443"`
`key="EnableSSLwithAuthlogicsServer"`	`value="true"`

(6) From an elevated admin command prompt run: `IISreset`

Diagnostics Logging

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\LoggingEnabled
```

Default Value: `0`

Accepted Values:

`0 = Disabled`

`1 = Enabled`

Used by components: Authlogics Server

Notes: When this value is enabled various log files will be created in the logging folder. These logs may be requested by a Authlogics support engineer.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\LoggingFolder
```

Default Value: `C:\Program Files\Authlogics Authentication Server\Log`

Used by components: Authlogics Server

Notes: This Value may be changed to an alternative valid local folder with the same NTFS permissions as the default folder.

Other settings

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\EnableSSLwithAuthlogicsServer
```

Default Value: `0`

Used by components: Authlogics Server; Management Console

Notes: When set to 1 all HTTP based requests on the *AuthlogicsPort* will be HTTPS/SSL based.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\AuthlogicsServerTimeout
```

Default Value: `30 (Decimal)`

Used by components: All

Notes: This value sets the timeout in seconds to be used when an Authlogics component establishes a connection to a resource.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\ProgramFolder
```

Default Value: `C:\Program Files\Authlogics Authentication Server`

Used by components: All

Notes: Changing this value is NOT supported.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication
Server\DirectoryServiceConnectionString
```

Default Value: Varies depending on directory service used.

Used by components: Authlogics Server; Management Console

Notes: Changing this value is NOT supported.

```
HKLM\SOFTWARE\Authlogics\Authlogics Authentication Server\DirectoryServiceType
```

Default Value: Varies depending on directory service used.

Used by components: Authlogics Server; Management Console

Notes: Changing this value is NOT supported.