ps-plugin-JWTTools

Utilities to create and validate JSON Web Tokens (JWT)

createtoken Function

Provides a custom function called `createtoken` that will generate a JWT. The function can take 8 parameters; however, only two are required.

Syntax

```
createtoken(externalSystemKey, sub, iss, alg, typ, aud, jti, exp)
```

Inputs

Input	Data Type	Required	Multiple	Description
externalSystemKey	Text	Yes	No	secure credential store key which contains field for appianPrivateKey and appianPrivateKeyPassword (privateKey used to sign token that is created). scs fields must be labeled exactly as described to be accessed by this function. Supports encrypted private keys in the PKCS#1 and PKCS#8 (RSA only) standard.
sub	Text	Yes	No	sub (Subject) - Generally an Enterprise or User ID
iss	Text	No	No	Claim - iss (Issuer)
alg	Text	No	No	Algorithm used to encrypt JWT - default RSA (RS256)
typ	Text	No	No	Type for the JWT Token - default JWT
aud	Text	No	No	Claim - aud (Audience)
jti	Text	No	No	Claim - jti (JWT ID)
exp	Number (Integer)	Yes	No	expiration time of generated token in ms (default 60000)

Returns

String with the JWT token

createtokenwithcustomclaims Function

Generates a JWT with custom claims and/or headers specified by the user. The function can take 11 parameters; however, only two are required.

Syntax

```
createtokenwithcustomclaims(externalSystemKey, sub, iss, alg, typ, aud, jti, exp, ver, claims, headers)
```

Inputs

Input	Data Type	Required	Multiple	Description
externalSystemKey	Text	Yes	No	secure credential store key which contains field for appianPrivateKey and appianPrivateKeyPassword (privateKey used to sign token that is created). scs fields must be labeled exactly as described to be accessed by this function. Supports encrypted private keys in the PKCS#1 and PKCS#8 (RSA only) standard.
sub	Text	Yes	No	sub (Subject) - Generally an Enterprise or User ID
iss	Text	No	No	Claim - iss (Issuer)
alg	Text	No	No	Algorithm used to encrypt JWT - default RSA (RS256)

Input	Data Type	Required	Multiple	Description
typ	Text	No	No	Type for the JWT Token - default JWT
aud	Text	No	No	Claim - aud (Audience)
jti	Text	No	No	Claim - jti (JWT ID)
exp	Number (Integer)	No	No	expiration time of generated token in ms (default 60000)
ver	Text	No	No	version field on the header
claims	Any Type	No	No	dictionary of custom claims to add to JWT. Does not support array custom claims, only supports String claims (i.e. no custom int or date claims)
headers	Any Type	No	No	dictionary of custom headers to add to JWT. Supports string and array claims)

Returns

String with the JWT token

createTokenWithSecretKey Function

Generates a JWT with a Secret Key.

Syntax

```
createTokenWithSecretKey(externalSystemKey, iss, alg, typ, aud, jti, exp, sub)
```

Inputs

Input	Data Type	Required	Multiple	Description
externalSystemKey	Text	Yes	No	secure credential store key which contains field for appianPrivateKey and appianPrivateKeyPassword (privateKey used to sign token that is created). scs fields must be labeled exactly as described to be accessed by this function
iss	Text	Yes	No	Claim - iss (Issuer)
alg	Text	Yes	No	Algorithm used to encrypt JWT - default RSA (RS256)
typ	Text	Yes	No	Type for the JWT Token - default JWT
aud	Text	No	No	Claim - aud (Audience)
jti	Text	No	No	Claim - jti (JWT ID)
exp	Number (Integer)	No	No	expiration time of generated token in ms (default 60000)
sub	Text	No	No	sub (Subject) - Generally an Enterprise or User ID

Returns

String with the JWT token

validateJWTSignature Function

Verify a received token against the issuing system's public key. Returns true if token was verified. Supports tokens encrypted with the RSA256 and RSA512 algorithm

Syntax

```
validatejwtsignature(externalSystemKey, token)
```

Inputs

Input	Data Type	Required	Multiple	Description
externalSystemKey	Text	Yes	No	secure credential store key which contains field for externalPublicKey (publicKey from token issuer used to verify token). scs fields must be labeled exactly as described to be accessed by this function
token	Text	Yes	No	Token to verify

Returns

Boolean

decodeJWT Function

Validate the signature of the received token using the received JWKS url, if the token is valid and not expired returns the decoded token, else return null. Implements a cache of publicKeys by keyId for every JWKS repository, the cache has a limit of 100 entries, the cache entries expire after 2 hours of last use

Syntax

```
decodeJWT(jwksUrl, token)
```

Inputs

Input	Data Type	Required	Multiple	Description
jwksUrl	Text	Yes	No	url of the JWKS repository
token	Text	Yes	No	Token to decode

Returns

Text the decoded token if signature is valid and the token is not expired

createDocuSignRSAJWTToken Function

Provides a custom function called `createdocusignrsajwttoken` that will generate a JWT that conforms to the specs of DocuSign. The function can take 8 parameters; however, only four are required.

Syntax

```
createdocusignrsajwttoken(externalSystemKey, iss, alg, typ, aud, scope, exp, sub)
```

Inputs

Input	Data Type	Required	Multiple	Description
externalSystemKey	Text	Yes	No	secure credential store key which contains field for appianPrivateKey and optional appianPrivateKeyPassword (privateKey used to sign token that is created). scs fields must be labeled exactly as described to be accessed by this function
iss	Text	Yes	No	Claim - iss (Issuer)
alg	Text	Yes	No	Algorithm used to encrypt JWT - use RS256 for DocuSign
typ	Text	Yes	No	Type for the JWT Token - use JWT for DocuSign
aud	Text	Yes	No	Claim - aud (Audience)
scope	Text	Yes	No	Claim - scope (signature for DocuSign)
	Number			

Input	Data Type	Required	Multiple	Description
exp	(Integer)	Yes	No	expiration time of generated token in ms (default 60000)
sub	Text	Yes	No	sub (Subject) - Generally an Enterprise or User ID

Returns

String with the JWT token

Example: Within an Interface or Expression Rule

```
rule!IntegrationRule(
    jwt: fn!createtoken(
    externalSystemKey: "jwt",
    iss: "Appian",
    exp: 60000
    ),
    onSuccess: {
        a!save(
            ri!accessCode,
            a!fromJson(fv!result.body).access_token
        )
    },
    onError: {
        a!save(
            ri!accessCode,
            fv!result
        )
    }
)
```

In this example, `rule!IntegrationRule` is an Integration which receives the JWT created by the plugin and returns an access token valid for 60 seconds.

##Previous Release Notes## ###1.4.1 Release Notes###

Security Updates. New function: `decodeJWT` - validate the signature of the received token using the received JWKS url