[#tokens-work] == How token encodings work

Tokens are objects that implement the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.IResolvable.html[IResolvable] interface, which contains a single resolve method. During synthesis, the {aws} CDK calls this method to produce the final value for tokens in your CloudFormation template.

= [NOTE]

You’ll rarely work directly with the IResolvable interface. You will most likely only see string-encoded versions of tokens.

====

[#tokens-work-types] === Token encoding types

Tokens participate in the synthesis process to produce arbitrary values of any type. Other functions typically only accept arguments of basic types, such as string or number. To use tokens in these cases, you can encode them into one of three types by using static methods on the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Token.html[cdk.Token] class.

* https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Token.html#static-aswbrstringvalue-options[Token.asString] to generate a string encoding (or call .toString() on the token object).
* https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Token.html#static-aswbrlistvalue-options[Token.asList] to generate a list encoding.
* https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Token.html#static-aswbrnumbervalue[Token.asNumber] to generate a numeric encoding.

These take an arbitrary value, which can be an IResolvable, and encode them into a primitive value of the indicated type.

= [IMPORTANT]

Because any one of the previous types can potentially be an encoded token, be careful when you parse or try to read their contents. For example, if you attempt to parse a string to extract a value from it, and the string is an encoded token, your parsing fails. Similarly, if you try to query the length of an array or perform math operations with a number, you must first verify that they aren’t encoded tokens.

====

[#tokens-check] == How to check for tokens in your app

To check whether a value has an unresolved token in it, call the https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.Token.html#static-iswbrunresolvedobj[Token.isUnresolved] (Python: is\_unresolved) method. The following is an example that checks if the value for our Amazon S3 bucket name is a token. If its not a token, we then validate the length of the bucket name:

==== [role=“tablist”] TypeScript:: + [source,javascript,subs=“verbatim,attributes”] — // …

export class CdkDemoAppStack extends cdk.Stack { constructor(scope: Construct, id: string, props?: cdk.StackProps) { super(scope, id, props);

…. // Define an S3 bucket const myBucket = new s3.Bucket(this, ‘myBucket’); // …

// Check if bucket name is a token. If not, check if length is less than 10 characters if (cdk.Token.isUnresolved(myBucketName)) { console.log(“Token identified.”); } else if (!cdk.Token.isUnresolved(myBucketName) && myBucketName.length > 10) { throw new Error(‘Maximum length for name is 10 characters.’); };

// … } } —- ….

JavaScript:: + [source,javascript,subs=“verbatim,attributes”] — const { Stack, Duration, Token, CfnOutput } = require(‘aws-cdk-lib’); // …

class CdkDemoAppStack extends Stack { constructor(scope, id, props) { super(scope, id, props);

…. // Define an S3 bucket const myBucket = new s3.Bucket(this, ‘myBucket’);

// …

// Check if bucket name is a token. If not, check if length is less than 10 characters if (Token.isUnresolved(myBucketName)) { console.log(“Token identified.”); } else if (!Token.isUnresolved(myBucketName) && myBucketName.length > 10) { throw new Error(‘Maximum length for name is 10 characters.’); };

// … } } —- ….

Python:: + [source,python,subs=“verbatim,attributes”] — from aws\_cdk import ( Stack, Token )

= …

class CdkDemoAppStack(Stack):

…. def **init**(self, scope: Construct, construct\_id: str, \*\*kwargs) -> None: super().\_\_init\_\_(scope, construct\_id, \*\*kwargs)

# Define an S3 bucket  
my\_bucket = s3.Bucket(self, "myBucket")  
  
# ...  
  
# Check if bucket name is a token. If not, check if length is less than 10 characters  
if Token.is\_unresolved(my\_bucket\_name):  
 print("Token identified.")  
elif not Token.is\_unresolved(my\_bucket\_name) and len(my\_bucket\_name) < 10:  
 raise ValueError("Maximum length for name is 10 characters.")  
  
# ... ----

….

Java:: + [source,java,subs=“verbatim,attributes”] — // … import software.amazon.awscdk.Token; import software.amazon.awscdk.services.s3.Bucket; // …

public class CdkDemoAppStack extends Stack { public CdkDemoAppStack(final Construct scope, final String id) { this(scope, id, null); }

…. public CdkDemoAppStack(final Construct scope, final String id, final StackProps props) { super(scope, id, props);

// Define an S3 bucket  
Bucket myBucket = Bucket.Builder.create(this, "myBucket")  
 .build();  
  
// ...  
  
// Get the bucket name  
String myBucketName = myBucket.getBucketName();  
  
// Check if the bucket name is a token. If not, check if length is less than 10 characters  
if (Token.isUnresolved(myBucketName)) {  
 System.out.println("Token identified.");  
} else if (!Token.isUnresolved(myBucketName) && myBucketName.length() > 10) {  
 throw new IllegalArgumentException("Maximum length for name is 10 characters.");  
}  
  
// ...

} } } —- ….

C#:: + [source,csharp,subs=“verbatim,attributes”] — using Amazon.CDK; using Constructs; using Amazon.CDK.AWS.S3; using Amazon.CDK.AWS.Lambda; using System; using System.Collections.Generic;

namespace CdkDemoApp { public class CdkDemoAppStack : Stack { internal CdkDemoAppStack(Construct scope, string id, IStackProps props = null) : base(scope, id, props) { // Define an S3 bucket var myBucket = new Bucket(this, “myBucket”);

…. // …

// Get the bucket name  
 var myBucketName = myBucket.BucketName;  
  
 // Check if bucket name is a token. If not, check if length is less than 10 characters  
 if (Token.IsUnresolved(myBucketName))  
 {  
 System.Console.WriteLine("Token identified.");  
 }  
 else if (!Token.IsUnresolved(myBucketName) && myBucketName.Length > 10)  
 {  
 throw new System.Exception("Maximum length for name is 10 characters.");  
 }  
  
 // ...  
}

} } —- ….

Go:: + [source,go,subs=“verbatim,attributes”] — // …

func NewCdkDemoAppStack(scope constructs.Construct, id string, props \*CdkDemoAppStackProps) awscdk.Stack { var sprops awscdk.StackProps if props != nil { sprops = props.StackProps } stack := awscdk.NewStack(scope, &id, &sprops)

…. // Define an S3 bucket myBucket := awss3.NewBucket(stack, jsii.String(“myBucket”), &awss3.BucketProps{})

// …

// Check if the bucket name is unresolved (a token) if tokenUnresolved := awscdk.Token\_IsUnresolved(myBucketName); tokenUnresolved != nil && *tokenUnresolved { fmt.Println(“Token identified.”) } else if tokenUnresolved != nil && !*tokenUnresolved && len(\*myBucketName) > 10 { panic(“Maximum length for name is 10 characters.”) }

// … } —- ==== ….

When we run cdk synth, myBucketName is identified as a token:

== [source,bash,subs=“verbatim,attributes”]

$ cdk synth –quiet Token identified. —

= [NOTE]

You can use token encodings to escape the type system. For example, you could string-encode a token that produces a number value at synthesis time. If you use these functions, it’s your responsibility to make sure that your template resolves to a usable state after synthesis.

====