:doctype: book

include::attributes.txt[]

// Attributes

[.topic] [#toolkit-library-configure] = Configuring your CDK Toolkit instance :info\_titleabbrev: Configure the CDK Toolkit :keywords: CDK Toolkit Library, Programmatic access, {aws} CDK, {aws} Cloud Development Kit ({aws} CDK), deploy, {aws} CloudFormation, Infrastructure as Code, synthesize

== [abstract]

## Learn how to customize your {aws} CDK Toolkit Library instance with options for message handling, {aws} profile selection, and stack selection strategies. This guide explains the available configuration options and how to implement them effectively to meet your specific deployment requirements.

// Content start

Learn how to customize your {aws} CDK Toolkit Library instance with options for message handling, {aws} profile selection, and stack selection strategies. This guide explains the available configuration options and how to implement them effectively to meet your specific deployment requirements.

[#toolkit-library-configure-profile] == Configuring your {aws} profile

When you use the CDK Toolkit Library, it makes API calls to {aws} using the SDK. While authentication is loaded automatically from your environment, you can explicitly specify which profile to use:

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

import { Toolkit } from ‘@aws-cdk/toolkit-lib’;

// Create a toolkit instance with a specific AWS profile const toolkit = new Toolkit({ sdkConfig: { profile: “my-profile” }, }); —

[#toolkit-library-configure-stacks] == Configuring stack selection

Most CDK Toolkit actions require you to specify which stacks to operate on. The +link:https://docs.aws.amazon.com/cdk/api/toolkit-lib/Package/toolkit-lib/Interface/StackSelector/[StackSelector]+ configuration controls this selection.

[#toolkit-library-configure-stacks-all] === Select all stacks

Use this when you want to operate on every stack in your CDK app:

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

import { StackSelectionStrategy } from ‘@aws-cdk/toolkit-lib’;

// Select all stacks in the cloud assembly await toolkit.deploy(cloudAssemblySource, { stacks: { strategy: StackSelectionStrategy.ALL\_STACKS } }); —

[#toolkit-library-configure-stacks-main] === Select only main assembly stacks

Use this to select only the top-level stacks from the main assembly:

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

// Select only top-level stacks await toolkit.deploy(cloudAssemblySource, { stacks: { strategy: StackSelectionStrategy.MAIN\_ASSEMBLY } }); —

[#toolkit-library-configure-stacks-single] === Select a single stack

Use this when your assembly contains exactly one stack and you want to assert this condition. If the assembly includes a single stack, it returns that stack. Otherwise, it throws an exception:

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

// Ensure there’s exactly one stack and select it await toolkit.deploy(cloudAssemblySource, { stacks: { strategy: StackSelectionStrategy.ONLY\_SINGLE } }); —

[#toolkit-library-configure-stacks-pattern] === Select stacks by pattern

Use this to select specific stacks by name pattern:

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

// Select stacks matching specific patterns await toolkit.deploy(cloudAssemblySource, { stacks: { strategy: StackSelectionStrategy.PATTERN\_MUST\_MATCH, patterns: [“Dev-\*“,”Test-Backend”], // Supports wildcards } }); —

= [TIP]

# Use PATTERN\_MUST\_MATCH\_SINGLE to ensure exactly one stack matches your patterns, or PATTERN\_MATCH if it’s acceptable for no stacks to match. Pattern matching supports wildcards like “\*” to match multiple stacks with similar names.

[#toolkit-library-configure-errors] == Configuring error handling

The CDK Toolkit uses structured errors to help you identify and handle issues. Each error includes:

* A *source* indicating where the error originated (toolkit or user).
* A specific *error type* (authentication, validation, etc.).
* A descriptive *message*.

[#toolkit-library-configure-errors-how] === Handling errors

Use the helper methods provided by the CDK Toolkit to detect and handle specific error types:

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

import { ToolkitError } from ‘@aws-cdk/toolkit-lib’;

try { // Attempt a CDK Toolkit operation await toolkit.deploy(cloudAssemblySource, { stacks: { strategy: StackSelectionStrategy.ALL\_STACKS } });

} catch (error) { // Handle specific error types if (ToolkitError.isAuthenticationError(error)) { // Example: AWS credentials are missing or invalid console.error(‘Authentication failed. Check your AWS credentials.’);

} else if (ToolkitError.isAssemblyError(error)) { // Example: Your CDK app has errors in stack definitions console.error(‘CDK app error:’, error.message);

} else if (ToolkitError.isDeploymentError(error)) { // Example: CloudFormation deployment failed console.error(‘Deployment failed:’, error.message);

} else if (ToolkitError.isToolkitError(error)) { // Handle all other Toolkit errors console.error(‘CDK Toolkit error:’, error.message);

} else { // Handle unexpected errors console.error(‘Unexpected error:’, error); } } —

= [IMPORTANT]

# Don’t rely on instanceof checks for error types, as they can behave unexpectedly when working with multiple copies of the same package. Always use the provided helper methods like ToolkitError.isAuthenticationError().

[#toolkit-library-configure-actions] == Configuring Toolkit actions

Each CDK Toolkit action (deploy, synth, list, etc.) has its own specific configuration options. These actions allow you to manage the complete lifecycle of your CDK infrastructure. For detailed information on configuring individual actions, see xref:toolkit-library-actions[Configure CDK Toolkit programmatic actions].

= [TIP]

# When building automation workflows, consider combining multiple actions in sequence. For example, you might want to synth your app, list the stacks to verify what will be deployed, and then deploy the infrastructure.