Generate C/C++ Code from MATLAB Code

Step 2 of 4 in Code Generation Guide: Generate Deployable C/C++ Code

After verifying MEX code behavior, generate standalone code for your project.

1. Specify input types.
2. Check for run-time issues.
3. Configure code generation build settings.
4. Generate standalone C/C++ code.
5. Understand generated code.

Specify Input Types

Before generating code, provide the input types to the code generator. The code generator then determines the data types to use in the generated code.

To automatically define input types, call your function by using example inputs or enter a script that calls your function in the prompt. Provide input types directly by providing example inputs. If your code requires a matrix of doubles of size 3-by-4, the example input can be `zeros(3,4)` or `ones(3,4)`.

For more information, see Input Type Specification for Code Generation.

Check for Run-Time Issues

After defining the input types for the code generator, perform an initial code generation and code execution to detect run-time errors that are harder to diagnose in the generated code.

1. To generate a MEX file for your code, click on the **Check for Issues** button.
2. You can open the Code Generation Report automatically by selecting the **Always create a code generation report** option in the **Debugging** section under **More Settings**.

See Check for Issues in MATLAB Code Using MEX Functions.

Configure Code Generation Build Settings

To create code according to your requirements, you can change the configuration settings of the code generator. In the **Generate Code** tab in the app, select the **More Settings** button at the bottom of the tab. This window lists the configuration settings that modify the generated code.

Use these settings to specify where the generated code should be built, apply target specific optimizations, enable variable-sizing support, include comments in the generated code, and apply other customizations for your generated code.

Generate Standalone C/C++ Code

After checking for run-time issues by generating a MEX file, generate standalone C/C++ code by choosing the required **Build type** under the **Generate Code** tab in the app.

To generate code for your project, click **Generate**.

Understand Generated Code

Access Code Generation Report

Use the code generation reports to view the generated C/C++ code, trace between the MATLAB® code and generated C/C++ code, and identify potential issues in the generated code.

After code generation, to open the code generation reports.

- In the MATLAB Coder™ app, in the **Debugging** settings, select the check box for **Always create a report** and **Automatically launch a report if one is generated**.

For more information, see Code Generation Reports.

Array Layout in Generated Code

Programming languages and environments typically assume a single-array layout for data. MATLAB uses column-major layout by default, whereas C and C++ use row-major layout.

To generate row-major code:

1. In the app, open the **Generate** dialog box. On the **Generate Code** page, click the **Generate** arrow.
2. Click **More Settings**.
3. On the **Memory** tab, set **Array layout** to `Row-major`.

See Code Design for Row-Major Array Layout.

Memory Allocation in Generated Code

For code generation, an array dimension is *fixed-size* or *variable-size*. If the code generator can determine the size of the dimension and that the size of the dimension does not change, then the dimension is fixed-size. When all dimensions of an array are fixed-size, the array is a *fixed-size* array.

You can generate code that allocates memory for fixed-size and variable-size arrays on the program stack or on the heap.

Static memory allocation allocates memory for arrays on the program stack at compile time. Static allocation is beneficial when:

- You know the upper bounds of all arrays in use.
- You have a large program stack.
- The arrays are small and take up less space on the program stack.

Dynamic memory allocation allocates memory on the heap dynamically at run time, instead of allocating memory statically on the stack. Dynamic memory allocation is beneficial when:

- You do not know the upper bound of an array.
- You do not want to allocate memory on the stack for large arrays.

Dynamic memory allocation can result in slower execution of the generated code. See Control Memory Allocation for Variable-Size Arrays.

To dynamically allocate memory for fixed-size and variable-size arrays in the generated code:

1. In the app, open the **Generate** dialog box. On the **Generate Code** page, click the **Generate** arrow.
2. Click **More Settings**.
3. On the **Memory** tab, select a value from the drop-down list for **Dynamic memory allocation for variable size arrays**.
4. On the **Memory** tab, select a value from the drop-down list for **Dynamic memory allocation for fixed size arrays**

Setting these options to `'Threshold'` causes arrays that are greater in size (in bytes) than the threshold value to be allocated dynamically.

Tips

Define Maximum Size for Unbounded Arrays

If you are generating code for arrays whose size depends on a user input, you can still set an upper limit for such inputs by using the `assert` function. For example:

```
function inSize(n)
assert(n < 25);
y = zeros(1,n);
end
```

When you cannot use dynamic memory allocation, define the upper bounds of arrays.

File I/O Support

The code generator includes limited support for functions like `coder.load`, `fread`, `fopen`, `fprintf`, and `fclose`.

Invoke Generated Code from Your C Project

The code generator provides an example main function for your reference when you generate static or dynamic libraries. See Use an Example C Main in an Application.

Generate Code at the Command Line

You can generate code and set all code generation options at the command line. See `codegen` and `coder.config`.

You can also use the code configuration objects to set these options at the command line. The options are properties of the configuration objects that are accessible by dot notation. See `coder.MexCodeConfig`, `coder.CodeConfig`, and `coder.EmbeddedCodeConfig`.

> **Note**
>
> To open a dialog box that includes the associated build configuration settings, double-click on the configuration object in the workspace.

Convert Your Project to Script

You can also convert your projects to a script by using the `-tocode` option of the `coder` command.

Optimize the Generated Code

While the code generator produces optimized code for most applications, you can generate efficient C/C++ code for your project by following some of these best practices:

- Pass arguments by reference
- Inline code
- Integrate optimized external code
- Disable run-time checks

For more information , see Optimize Generated C/C++ and MEX Code.

Create Reports at the Command Line

When generating code at the command line, use these `codegen` options:

- To generate a report, use the `-report` option.
- To generate and open a report, use the `-launchreport` option.

Alternatively, use configuration object properties:

- To generate a report, set `GenerateReport` to `true`.
- If you want the `codegen` command to open the report for you, set `LaunchReport` to `true`.

See Also

`coder.cstructname` | `coder.MexCodeConfig` | `coder.CodeConfig` | `coder.EmbeddedCodeConfig` | `coder.load` | `fread` | `fclose` | `fprintf` | `fopen` | `coder.opaque`

Related Topics

- Specify Objects as Inputs at the Command Line
- Code Design for Row-Major Array Layout
- Control Memory Allocation for Variable-Size Arrays

Code Generation Guide: Generate Deployable C/C++ Code NEXT ›