ARC Grid Construction Guide

Overview

This document outlines the procedures for constructing computational grids in the A.R.C. (Adaptive Reactive Code) framework. Before constructing a grid, it is essential to understand the available options. The table below lists the functions used to generate grids, their input arguments, the structures they produce, and the key properties of those structures.

Grid Construction Functions

Function	**Input**	**Structure**	**Properties**
`Construct1DCartesian`	`domain_length`, `zones`, `ghost_zones`, `grid_center`, `units`	`CartesianGrid1D`	`xcoord::UniformAxis` `bounds::Tuple{Float64, Float64}` `ghost_bounds::Tuple{Float64, Float64}` `units::String`
`Construct2DCartesian`	`xdomain_length`, `ydomain_length`, `xzones`, `yzones`, `ghost_zones`, `grid_center`, `units`	`CartesianGrid2D`	`xcoord::UniformAxis`, `ycoord::UniformAxis` `xbounds`, `ybounds` `xghost_bounds`, `yghost_bounds` `cell_areas::Vector{Float64}` `units::String`
`Construct1DSpherical`	`domain_length`, `zones`, `ghost_zones`, `grid_center`, `units`	`SphericalGrid1D`	`rcoord::UniformAxis` `bounds`, `ghost_bounds` `units::String`
`Construct2DSpherical`	`domain_length`, `zones`, `theta_zones`, `ghost_zones`, `grid_center`, `units`	`SphericalGrid2D`	`rcoord::UniformAxis`, `thetacoord::UniformAxis` `rbounds`, `thetabounds` `rghost_bounds`, `thetaghost_bounds` `cell_areas::Vector{Float64}` `units::String`
`ConstructUniformAxis`	`domain_length`, `zones`, `ghost_zones`, `grid_center`, `coord_system`	`UniformAxis`	`centers`, `interfaces`, `all_centers`, `all_interfaces` `spacing`, `total_zones` `zones`, `ghost_zones` `coord_system`

Usage

To construct a grid in A.R.C., you call one of the four main constructor functions depending on the dimensionality and coordinate system required. Internally, these functions utilize `ConstructUniformAxis`, which serves as the foundational routine for generating uniformly spaced coordinate axes. Each returned structure is designed for intuitive and minimal access to key properties. These structures promote ease of use and efficient memory access patterns, similar in style to C-like structured programming. An example of creating a 2D Cartesian grid follows: Once the ARC package is loaded into Julia, type the following into the Julia REPL

```
Construct2DCartesian(1.0, 1.0, 10, 10, 3, 0.0, :cartesian, "cm")
```

This creates a two dimensional cartesian grid with end points -0.5, and 0.5 for both x and y coordinates. It divides the x axis into 10 zones and the y axis into 10 zones, adding 3 ghost zones on the ends of each axis. This means that the total zones in this grid are 10 (for x axis) $*$ 10 (for y axis) $+$(2 (one set of ghost zones for each end) $*$ ghost zones)^2 $=$ 136 zones in total.

If you define this as a variable say

```
xyGrid = Construct2DCartesian(1.0, 1.0, 10, 10, 3, 0.0, :cartesian, "cm")
```

then we can access the individual properties of our newly built grid. Referencing the above table, we see that the properties of a Cartesian2Dgrid allow us to use the xyGrid variable to access these values.

For example,

```
println(xyGrid.xcoord.centers)
println(xyGrid.xcoord.interfaces)
println(xyGrid.xbounds)
println(xyGrid.ybounds)
```

These commands will print the grid centers, interfaces, and display the x and y bounds. The output should look like this...

```
ARC.CartesianGrid2D(ARC.UniformAxis([-0.5, -0.38888888888888884, -0.2777777777777778, -0.1666666666666667,
-0.05555555555555558, 0.05555555555555558, 0.16666666666666663, 0.2777777777777778, 0.38888888888888884,
0.5], [-0.55, -0.44000000000000006, -0.33000000000000007, -0.22, -0.10999999999999999, 0.0, 0.10999999999999999,
0.21999999999999997, 0.33000000000000007, 0.44000000000000006, 0.55], [-0.8, -0.6933333333333334, -0.5866666666666667,
-0.4800000000000001, -0.3733333333333334, -0.2666666666666668, -0.15999999999999992, -0.053333333333333344,
0.053333333333333344, 0.15999999999999992, 0.2666666666666666, 0.3733333333333333, 0.48000000000000015,
0.5866666666666667, 0.6933333333333334, 0.8], [-0.8500000000000001, -0.7437500000000001, -0.6375000000000002,
-0.53125, -0.42500000000000004, -0.3187500000000001, -0.21249999999999997, -0.10624999999999996, 0.0,
0.10624999999999996, 0.21249999999999997, 0.3187500000000001, 0.42500000000000004, 0.53125, 0.6375000000000002,
0.7437500000000001, 0.8500000000000001], 0.1, 16, 10, 3, :cartesian), ARC.UniformAxis([-0.5, -0.38888888888888884,
-0.2777777777777778, -0.1666666666666667, -0.05555555555555558, 0.05555555555555558, 0.16666666666666663,
0.2777777777777778, 0.38888888888888884, 0.5], [-0.55, -0.44000000000000006, -0.33000000000000007, -0.22,
-0.10999999999999999, 0.0, 0.10999999999999999, 0.21999999999999997, 0.33000000000000007, 0.44000000000000006,
0.55], [-0.8, -0.6933333333333334, -0.5866666666666667, -0.4800000000000001, -0.3733333333333334, -0.2666666666666668,
-0.15999999999999992, -0.053333333333333344, 0.053333333333333344, 0.15999999999999992, 0.2666666666666666,
0.3733333333333333, 0.48000000000000015, 0.5866666666666667, 0.6933333333333334, 0.8], [-0.8500000000000001,
-0.7437500000000001, -0.6375000000000002, -0.53125, -0.42500000000000004, -0.3187500000000001, -0.21249999999999997,
-0.10624999999999996, 0.0, 0.10624999999999996, 0.21249999999999997, 0.3187500000000001, 0.42500000000000004,
0.53125, 0.6375000000000002, 0.7437500000000001, 0.8500000000000001], 0.1, 16, 10, 3, :cartesian), (-0.5,
0.5), (-0.5, 0.5), (-0.8, 0.8), (-0.8, 0.8), 0.0, [0.010000000000000002 0.010000000000000002 ... 0.010000000000000002
0.010000000000000002; 0.010000000000000002 0.010000000000000002 ... 0.010000000000000002 0.010000000000000002;
... ; 0.010000000000000002 0.010000000000000002 ... 0.010000000000000002 0.010000000000000002; 0.010000000000000002
0.010000000000000002 ... 0.010000000000000002 0.010000000000000002], ”cm”)
```