AN2504 – Controlling Radar Beams with the SGE

January 2025

Battlespace Simulations, Inc.

8305 Catamaran Circle

Lakewood Ranch, FL 34202

If you have questions or comments, please contact us at support@bssim.com.

Overview

One of the most powerful aspects of MACE and the MACE API is the physics based modeling we do for all signals in the battlespace, throughout the EM spectrum from IR signals to UV signals. The SGE is extremely powerful, and gives you the developer the ability to change modify any signal in the environment down to the pulse level.

In this application note, we're going to break down a common user request – controlling radar beams from a plugin, codescript, or command.

The full source code discussed in this application note can be found in the `CueTargetTracker.cs` codescript destributed w/ MACE.

What is a beam?

A radar beam is the focused emission of electromagnetic waves produced by a radar system's antenna. It is used to detect and track objects within its coverage area. The beam typically has a conical or fan-like shape and is formed by the directional radiation pattern of the antenna, which concentrates energy in specific directions while minimizing energy in others.

Key Characteristics:

- **Frequency:** Radar beams operate at specific frequencies, typically in the microwave or radio wave spectrum, to achieve desired detection capabilities.
- **Beamwidth:** This refers to the angular spread of the radar beam, which determines the system's resolution and coverage. Narrower beams provide higher resolution but cover smaller areas.
- **Polarization:** The orientation of the electromagnetic wave's electric field, which can be vertical, horizontal, circular, or elliptical, affects the interaction with targets.
- **Range:** The radar beam travels outward from the antenna, reflecting off objects (targets) and returning to the receiver for analysis.
- **Directionality:** Radar beams can be steered mechanically (by moving the antenna) or electronically (using phased arrays).

The radar beam's characteristics are critical for its ability to detect, locate, and identify objects such as aircraft, ships, or weather phenomena

In the SGE, beams are associated with Modes, which are in turn associated with Emitters and, ultimately, Devices. Devices define the physical characteristics of the the antenna, its physical limits, etc. Beams in the SGE can be fixed and moved when the antenna is

physically moved, or beams can be moved by being associated with electronic scan patterns. In this case, we don't want to change the shape or other physical parameters of the beam – we just want to tell it where to point. That will be dictated by the `ScanAngles` property of our desired mode, so we really just need to override those. Fortunately there's an SGE event that will permit us to do just.

Subscribing to SGE Events

When changing the default behaviour of how an SGE device works, it's best to do so when the SGE is done with all of the internal processing for the device. There is a global event that is raised when the SGE is done processing all SGE devices, the `CycleComplete` event, and you can subscribe to it as follows:

```
SGE.IEWEnvironment.CycleComplete += HandleEWCycleComplete;
```

The `HandleEWCycleComplete` event is our local delegate/handler for the cycle complete event.

The SGE processes all devices and emitters in a mission in fixed, 100ms time blocks. Any emitters it doesn't get to in a 100ms time block are left as incomplete. This is normally not a problem in a typical MACE mission, but you can get an idea of how much processing MACE is doing in your mission in the MACE performance window. We can change a device's behavior by overriding the data it just computed its computation cycle with our own data. To do that, we'll first get a reference to the radar mode we care about, and after each SGE processing cycle is complete, we'll compute a new set of scan angles for the mode that tell it where to point its beam.

Getting a reference to the radar mode

To get a reference to the target tracking mode for an entity's radar, you would do something like the following:

```
IPhysicalEntity _ownship = MissionInstance.Mission.Map.SelectedEntity;
List<IEquipment> radars = _ownship.GetEquipmentList(CapabilityEnum.Radar);

foreach(IEquipment radar in radars)
{
    IEquipmentEW ew = radar as IEquipmentEW;
    if (ew != null)
    {
        foreach (IMode m in ew.Device.Emitter.Modes)
        {
            if (m.Function == ModeFunctionType.Tracking || m.Function ==
ModeFunctionType.AcquisitionTracking)
            {
```

```
                _mode = m;
                _mission.LogMissionEvent($"TT Codescript: Target tracking
mode found on device {ew.Device.Name}");
            }
        }
    }
}
```

In the above snippet, we first get a list of all radar devices from the entity's equipment list. We then iterate through each equipment item and find a mode with a function type of either `Tracking` or `AcquisitionTracking`. You can be more specific and filter by parameter, or even name.

We save a reference to our tracking mode, if we find one, in a globally defined/scoped instance of `IMode` called `_mode`. Our cycle complete handler will use that reference to update where the radar beam is pointed. In this case, we'll have the beam "stare" at an entity we've defined as a "target" entity (from a user mouse click elsewhere in the codescript). Let's take a look at the handler to see how it does that:

```
private static void HandleEWCycleComplete(object Sender, EventArgs e)
{
    try
    {
        if (_targetEntity != null)
        {
            // update scan angles to have target tracker stare at target
            RangeAngle ra =
Geographic.GetRelativePosition(_ownship.Position.Geographic,
_targetEntity.Position.Geographic);

            ScanAngles sa = new ScanAngles(SGE.MissionTime);

            for (int i = 0; i < 100; i++)
            {
                sa.AzimuthAngles[i] = ra.Azimuth;
                sa.ElevationAngles[i] = ra.Elevation;
                sa.Amplitudes[i] = 1.0;
            }
            _mode.ScanAngles = sa;
        }
    }
    catch (Exception ex)
    {
        MissionInstance.Mission.Logger.ErrorMessage(ex);
    }
}
```

We first calculate the pointing angles between our `_ownship` instance and a target entity, if it exists, in world space using a `RangeAngle` object to hold the result. We ten define a new `ScanAngles` object; a `ScanAngles` object contains 100 ms worth of scan data that the SGE will utilize in its next update cycle. We could define complex scan patterns if we wanted to, but since we just want to have our beam stare at the target location, we simply fill out the `AzimuthAngles`, `ElevationAngles`, and `Amplitudes` collections w/ the `RangeAngle` results. We then assign the `ScanAngles` object we just defined to the `ScanAngles` property for our tracking mode.

"But wait a second!", some of you protest. "Won't the target move in that 100 ms time block?". It sure will, so your pointing angles won't be exactly correct for each 1ms time slice in the 100ms processing block. We're updating at 10 Hz, however, which for the purposes of this demonstration is close enough. If you needed higher fidelity, you could certainly dead reckon out the expected position of the target entity for the next 100ms if you need to.

Conclusion

That's really all there is to it, but hopefully you can see just how powerful the SGE is, and we hope this gives you some ideas on how you can make good use of it in your own plugins, codescripts, and commands!