AN2502 – Listening to Signals in the SGE

A BSI Application Note

January 2025

Battlespace Simulations

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 – listening for signals on a Radar Warning Receiver (RWR).

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

Devices, Emitters, and Modes

In the context of MACE and the SGE, Device, Emitter, and Mode are distinct concepts that relate to the Signal Generation Engine (SGE) and its role in simulating electromagnetic environments. Here's a breakdown:

Device

A Device in MACE refers to any object that can emit, receive, or process electromagnetic signals. Examples include radars, radios, jammers, and IR detectors. Devices are components of entities (platforms or equipment in the simulation) and are responsible for interacting with the electromagnetic environment as modeled by the SGE. Devices manage interactions like scanning, emitting signals, or processing received signals, and they often interface with other simulation systems like Track Processors or AI subsystems.

Emitter

An Emitter is a component of a Device responsible for generating electromagnetic signals. For example, a radar or a communication system will have an emitter that transmits signals into the environment. Emitters have specific parameters, such as frequency, power, beam pattern, and modulation, which define their behavior. In the SGE, Emitters are configured to model real-world behaviors, including scanning patterns and emission timing. These can also include characteristics like pulse repetition frequency (PRF) and antenna gain.

Mode

A Mode defines the operational state or configuration of an Emitter. For instance, a radar might have different modes for search, track, or acquisition. Modes determine how an Emitter behaves in specific scenarios, including the type of signals it emits, the pulse or waveform characteristics, and the target detection parameters. Modes often correlate with mission-specific requirements or environmental conditions, allowing dynamic adjustments during the simulation. For example, a radar's low-power mode might be used for stealth operations, while high-power modes are employed for extended range.

For an RWR, each platform with an RWR will have a RWR as an equipment item. That equipment will have an RWR, which in turn has an RWR receiver, and the receiver will have a mode that defines the operating parameters for receiver. Equipment with SGE devices attached will implement the `IEquipmentEW` interface. Different kinds of devices will implement different interfaces based on the type of device it is. In the case of a RWR device, it will implement the `IDeviceReceiver` interface.

With those definitions and knowledge of the basic framework in hand, let's take a look at the code:

```
if (_mission.Map.SelectedEntity != null)
{
    _ownship = _mission.Map.SelectedEntity;

    List<IEquipment> ewsEquipment = _ownship.EquipmentList.Where(eq => eq
is IEquipmentEW).ToList();
    foreach (IEquipment eq in ewsEquipment)
    {
        IEquipmentEW ew = eq as IEquipmentEW;
        if (ew != null)
        {
            _rwr = ew.Device.Is<IDeviceReceiver>();
            if (_rwr != null)
            {
                _mission.LogMissionEvent($"Device: {_rwr.Name}");
                _rwr.IFSignalDataStreamThreadPool += HandleRwrSignalData;
            }
        }
    }
}
```

Using a local reference to the mission, we're attempting to get a reference to the user selected entity. Our set a local reference to that (`_ownship`), then iterate through that entity's equipment list. If the equipment has a device, it will implement the `IEquipmentEW` interface; we simply check to see if our equipment implements that interface, and if it does, we then attempt to set a local reference to the RWR by checking to see in the device implements `IDeviceReceiver`. If it does, we then subscribe to the signals received by that device by subscribing to the `IFSignalDataStreamThreadPool`.

That signal data handler looks something like the following:

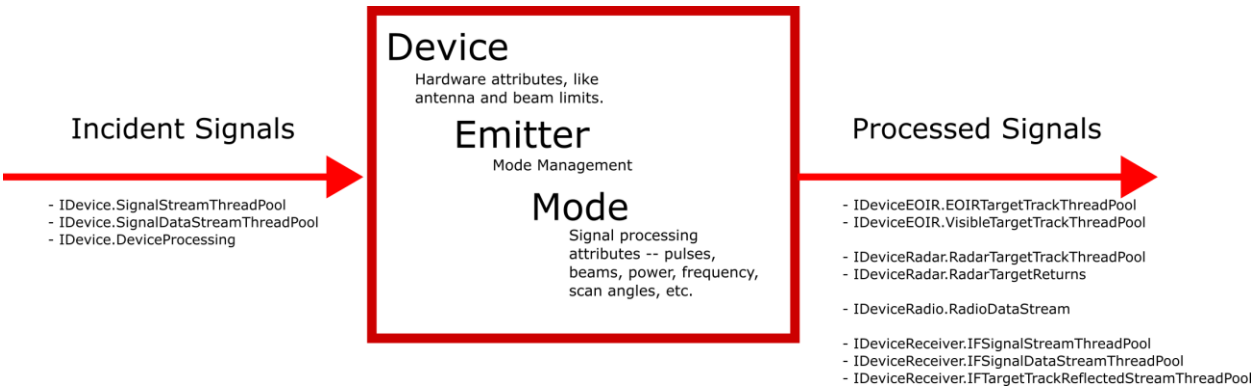
```
public static void HandleRwrSignalData(object sender, SignalDataEventArgs
args)
{
    foreach (SignalData sd in args.SignalDataList)
    {
        IPhysicalEntity ipe = sd.TargetEntity.PlatformTag as
IPhysicalEntity;
        if (ipe != null)
```

```
            {
                _mission.LogMissionEvent($"RWR Target Detected: {ipe.Type}");
            }
        }
    }
```

In the signal data handler, we can iterate through each of the `SignalData` objects available in the event arguments and log a mission event w/ the MACE type of the platform represented by the `TargetEntity` in the signal data object – in other words, originator of the signal in question. *Note that this is just for illustration purposes*; as SGE events can potentially raise many hundreds of events per frame, logging like this for SGE events could have deliterious performance implications.

Types of Signal Events

Signal related events are hosted at the Device level and can generally be categorized as either "pre-processing" events or "post-processing" events. "Pre-processing" events represent signals received by the antenna that exceed the minimum detectable signal threshold for the device. "Post-processing" events represent the signals after processing by the device, taking into account factors such as band pass limits, tuned frequency, and receiver sensitivity.



To learn more about how MACE models the EW environment, please see "MACE Electromagnetic Processing", available via the MACE Quick menu, or at its default install location of `C:\Users\Public\Documents\MACE\documentation\EW`.

What is a Thread Pool event?

A thread pool event refers to a task or unit of work submitted to the .NET ThreadPool, which is managed by the CLR (Common Language Runtime). The .NET ThreadPool provides a way to execute tasks on background threads efficiently without manually creating and managing threads. They also prevent you from having to marshall the results of thread from one context to another.

Events on `IDevice`

- **`SignalStream`** – Event that fires periodically returning signal data received in the vicinity of this device. Returns a list of `SGE.Signal` type suitable for generating

emitter audio and virtual display signals. Data returned is filtered by signal power at the device location in world space using the `Device.MinimumDetectableSignal` as a signal threshold; it represents a "pre-receiver" signal and does not filter by sensitivity or frequency. This event returns on the thread context that registered for this event. Not recommended for new development – use `SignalStreamThreadPool` instead.

- **`SignalStreamThreadPool`** – Event that fires periodically returning signals received in the vicinity of this device. Returns a list of `SGE.Signal` type suitable for generating emitter audio and virtual display signals. Data returned is filtered by signal power at the device location in world space using the `Device.MinimumDetectableSignal` as a signal threshold. This event returns on a thread pool thread.

- **`ProcessedSignalStream`** – Event that fires periodically returning signals passed by an external receiver simulation. Returns a list of `SGE.Signal` type suitable for generating emitter audio and virtual display signals. Data returned is filtered by an external receiver simulation that passes these signals and fired this event. This event returns on a thread determined by the external simulation that fired the event.

- **`SignalDataStreamThreadPool`** – Event that fires periodically returning signal data received in the vicinity of this device. Returns a list of `SGE.SignalData` type suitable for constructive device detection of signals. Data returned is filtered by the signal power at the device position and the `Device.MinimumDetectableSignal`. This event returns on a thread pool thread.

- **`DeviceProcessing`** – Event that fires periodically during processing cycle of all other signal type events. This event is an empty processing function event that allow users to implement custom signal processing in the same parallel for each loop that all other processing functions of the device use. Note that excessive processing time will directly affect the performance of SGE. Any exception thrown by the handler function will result in all subscribers being unsubscribed.

Events on `IDeviceEOIR`

- **`EOIRTargetTrack`** – Event that fires periodically returning target tracks in range of this device. Returns a list of `SGE.TargetTrack` type suitable for constructive simulation. Data returned is filtered by ReceiverData for a given target at the device position and target in FOV some time during this period. This event returns on the thread context that registered for this event. Not recommended for new development – use `EOIRTargetTrackThreadPool` instead.

- **`EOIRTargetTrackThreadPool`** – Event that fires periodically returning target tracks in range of this device. Returns a list of `SGE.TargetTrack` type suitable for constructive simulation. Data returned is filtered by ReceiverData for a given target at the device position and target in FOV some time during this period. This event returns on a thread pool thread.

- **`VisibleTargetTrack`** – Event that fires periodically returning target tracks in range of this device. Returns a list of `SGE.TargetTrack` type suitable for constructive simulation. Data returned is filtered by ReceiverData for a given target at the device position and target in FOV some time during this period. This event returns on the thread context that registered for this event. Not recommended for new development – use `VisibleTargetTrackThreadPool` instead.

- **`VisibleTargetTrackThreadPool`** – Event that fires periodically returning target tracks in range. Returns a list of `SGE.TargetTrack` type suitable for constructive simulation. Data returned is filtered by ReceiverData for a given target at the device position and target in FOV some time during this period. This event returns on a thread pool thread.

Events on `IDeviceRadar`

- **`RadarTargetTrack`** – Event that fires periodically returning target tracks in range of this device. Returns a list of `SGE.TargetTrack` type (including received jammer signals) suitable for constructive radar simulation. Data returned is filtered by the radar range equation (using `IMode.ReceiverData`) for a given target at the device location in world space and target in 3dB beamwidth of radar beam some time during this period. This event returns on the thread context that registered for this event. Not recommended for new development – use `RadarTargetTrackThreadPool` instead.

- **`RadarTargetTrackThreadPool`** – Event that fires periodically returning target tracks in range of this device. Returns a list of `SGE.TargetTrack` type (including received jammer signals) suitable for constructive radar simulation. Data returned is filtered by the radar range equation (using `IMode.ReceiverData`) for a given target at the device location in world space and target in 3dB beamwidth of radar beam some time during this period. This event returns on a thread pool thread.

- **`RadarTargetReturns`** – Event that fires periodically returning targets in range of this device. Returns a list of `SGE.TargetReturn` type suitable for virtual radar displays. Data returned is filtered by target return power at the device location in world space, `Constants.MinRxNoiseInWatts`, and the `DeviceRadar.MaxEffectiveRange`. This event returns on the thread context that registered for this event.

Events on `IDeviceRadio`

- **`RadioDataStream`** – Event that fires periodically returning signal data received by this device. Returns a list of `SGE.SignalData` type suitable for virtual and constructive device detection of radio signals. Data returned is filtered by the diffracted signal power at the device location in world space, `IMode.ReceiverData.Sensitivity` and radio's tuned frequency. This event returns on a thread pool thread.

Events on `IDeviceReceiver`

- **`IFSignalStream`** - Returns a list of `SGE.Signal` type suitable for generating emitter audio and virtual display signals. Data returned is filtered by band pass and sensitivity of the receiver mode. This event returns on the thread context that registered for the event. Not recommended for new development – use `IFSignalStreamThreadPool` instead.

- **`IFSignalStreamThreadPool`** – Returns a list of `SGE.Signal` type suitable for generating emitter audio and virtual display signals. Data returned is filtered by band pass and sensitivity of the receiver mode. This event returns on a thread pool thread.

- **`IFSignalDataStream`** – Returns a list of `SGE.SignalData` type suitable for constructive device detection of signals. Data returned is filtered by band pass and sensitivity of the receiver mode. This event returns on the thread context that registered for the event. Not recommended for new development – use `IFSignalDataStreamThreadPool` instead.

- **`IFSignalDataStreamThreadPool`** – Returns a list of `SGE.SignalData` type suitable for constructive device detection of signals. Data returned is filtered by band pass and sensitivity of the receiver mode. This event returns on a thread pool thread.

- **`IFTargetTrackReflectedStreamThreadPool`** – Event that fires periodically returning target track data received by a device by both direct path and reflected backscatter path. Returns a list of `SGE.TargetTrack` type suitable for constructive and virtual device detection of signals. Data returned is filtered by band pass and sensitivity of the receiver mode. This event returns on a thread pool thread.