

# 威胁源仿真库文档

欢迎使用威胁源仿真库文档。

## 简介

威胁源仿真库是一个基于 **.NET 8.0** 的高性能仿真类库，用于模拟和仿真各种威胁源。它提供了以下主要功能：

- **导弹仿真**：支持多种制导系统（激光、红外、毫米波等）
- **目标仿真**：各类设备和目标的建模
- **传感器仿真**：雷达、光电等传感器系统
- **制导系统仿真**：完整的制导算法实现
- **事件驱动架构**：高效的仿真事件系统
- **智能数据管理**：TOML配置文件支持和智能路径解析
- **外部系统集成**：支持Unity、虚幻引擎等第三方引擎集成

## 系统要求

### 基本要求

- **.NET 8.0** 或更高版本
- Windows、Linux 或 macOS 操作系统
- Visual Studio 2019+ 或 Visual Studio Code（推荐）

### C# 开发

- .NET 8.0 SDK
- 支持 .NET 8.0 的 IDE

### C++ 开发

- Windows 操作系统
- Visual Studio 2019+（支持C++/CLI）
- .NET 8.0 运行时

## 支持的开发语言

- **C# (.NET)**：原生支持，提供完整的API和功能
- **C++**：通过C++/CLI包装层支持，可以在C++项目中使用全部功能

## 快速开始

请参阅[入门指南](#)了解如何使用该库。该指南包含了C#和C++两种语言的使用示例。

## API文档

完整的API文档请参阅[API参考](#)。

## 示例代码

## 基础仿真示例

- [仿真示例说明](#) - 基础仿真功能的详细说明
- [C# 红外成像制导导弹仿真](#) - C# 仿真示例
- [C++ 红外成像制导导弹仿真](#) - C++/CLI 仿真示例

## 第三方引擎集成示例

- [集成示例说明](#) - 第三方引擎集成的详细说明
- [虚幻引擎集成示例](#) - 虚幻引擎集成示例代码
- [Unity引擎集成示例](#) - Unity引擎集成示例代码

# 示例功能说明

## 基础仿真示例

这些示例展示了如何使用ThreatSource进行基础仿真：

1. **C# 仿真示例**
  - 使用SimulationManager管理仿真
  - 从TOML配置文件创建导弹实体
  - 设置目标和仿真环境
  - 订阅和处理仿真事件
  - 运行仿真循环并获取结果
2. **C++/CLI 仿真示例**
  - C++/CLI包装层的使用方法
  - 托管和非托管代码的交互
  - 事件处理和资源管理
  - 跨语言的仿真集成

## 第三方引擎集成示例

这些示例展示了如何将ThreatSource与游戏引擎集成：

1. **虚幻引擎集成示例**
  - ThreatSource与虚幻引擎的双向通信
  - Actor与仿真实体的映射和同步
  - 坐标系转换（右手系到左手系）
  - 事件驱动的视觉效果处理
2. **Unity引擎集成示例**
  - ThreatSource与Unity引擎的双向通信
  - GameObject与仿真实体的映射和转换
  - MonoBehaviour生命周期管理
  - 实时状态同步和视觉效果

# 核心特性

## 事件驱动架构

- 类型安全的事件系统
- 异步事件处理
- 异常隔离机制

## 智能数据管理

- TOML配置文件支持
- 智能路径解析
- 复合制导配置后处理

## 高性能仿真

- 优化的运动学计算
- 高效的碰撞检测
- 可配置的仿真精度

## 扩展性设计

- 模块化架构
- 插件式制导系统
- 灵活的实体系统

所有示例代码都提供了详细的注释和说明，可以作为实际开发的参考。

# 导弹工作原理参考手册

版本：1.1.22

本文档详细介绍威胁源库中实现的各种导弹制导系统的工作原理和技术特点。

## 1. 激光半主动制导导弹

### 1.1 工作原理

激光半主动制导导弹通过接收目标反射的激光能量进行制导，是一种精确制导武器系统。

### 1.2 工作阶段

#### 1. 发射阶段

- 导弹发射，发动机点火
- 初始化制导系统
- 准备接收激光照射信号

#### 2. 巡航阶段

- 导弹按预定轨迹飞行
- 制导系统处于待机状态
- 等待激光指示器照射目标

#### 3. 制导阶段

- 激光指示器开始照射目标
- 四象限探测器接收反射激光
- 计算光斑偏移量
- 生成制导指令修正飞行轨迹

#### 4. 攻击阶段

- 导弹接近目标
- 引爆战斗部
- 完成攻击任务

### 1.3 关键技术

#### 1. 四象限探测器

- 精确测量光斑位置偏移
- 提供水平和垂直误差信号
- 灵敏度可调节

#### 2. 激光编码识别

- 支持多种编码方式
- 防止敌方激光干扰
- 确保制导信号可靠性

#### 3. 比例导引控制

- 使用比例导引律计算制导加速度
- 加速度平滑处理，减少突变
- 最大加速度限制保护

## 1.4 技术特点

- **制导精度高**：四象限探测器提供高精度角度测量
- **抗干扰能力强**：激光编码技术防止干扰
- **全天候作战**：不受天气影响（轻微衰减）
- **需要持续照射**：激光指示器必须持续照射目标

# 2. 激光驾束制导导弹

## 2.1 工作原理

激光驾束制导导弹沿着激光束的中心线飞行，通过检测与激光束的偏差进行轨迹修正。

## 2.2 工作阶段

### 1. 发射阶段

- 导弹发射进入激光束
- 激光束探测器开始工作
- 建立初始制导基准

### 2. 驾束飞行阶段

- 持续检测与激光束中心线的偏差
- PID控制器计算修正指令
- 实时调整飞行轨迹

### 3. 末段制导阶段

- 接近目标区域
- 提高制导精度
- 准备攻击

## 2.3 关键技术

### 1. 激光束偏差检测

- 计算导弹与激光束中心线的距离
- 实时监测偏差变化
- 提供三维位置修正信号

### 2. PID控制系统

- 比例控制：响应当前偏差
- 积分控制：消除稳态误差
- 微分控制：预测偏差变化趋势

### 3. 非线性增益控制

- 根据偏差大小调整控制增益
- 大偏差时快速响应
- 小偏差时精确控制

## 2.4 技术特点

- **轨迹可控**：严格按照激光束路径飞行
- **精度极高**：PID控制提供精确轨迹跟踪
- **实时响应**：快速响应轨迹偏差
- **需要连续制导**：激光束必须持续到命中

# 3. 红外指令制导导弹

## 3.1 工作原理

红外指令制导系统由导弹和地面红外测角仪组成，通过红外信号建立通信链路，实现制导控制。

## 3.2 工作阶段

#### 1. 红外信号建立阶段

- 导弹发射后点亮红外热源
- 红外测角仪探测导弹红外信号
- 建立跟踪链路

#### 2. 双目标跟踪阶段

- 红外测角仪同时跟踪导弹和目标
- 计算两者的相对位置关系
- 生成制导指令

#### 3. 指令传输阶段

- 测角仪通过事件系统发送制导指令
- 导弹接收并解析指令
- 更新飞行轨迹

#### 4. 制导执行阶段

- 计算期望飞行方向
- 应用转向速率平滑
- 执行轨迹修正

### 3.3 关键技术

#### 1. 红外测角仪跟踪

- 具备一定的最大跟踪距离
- 具有较大的视场角
- 高精度角度测量
- 高频率更新

#### 2. 制导指令计算

- 计算测角仪到导弹的向量
- 计算测角仪到目标的向量
- 生成三维制导指令

#### 3. 导弹制导控制

- 转向速率平滑处理
- 提前量计算
- 自适应制导参数调整

### 3.4 技术特点

- 远程制导：地面测角仪提供远程制导能力
- 双目标跟踪：同时跟踪导弹和目标
- 指令制导：实时传输制导指令
- 需要通信链路：依赖红外通信链路

## 4. 红外成像末制导导弹

### 4.1 工作原理

红外成像末制导导弹通过红外成像探测器实现目标的自动探测、识别和跟踪，具备完全自主的制导能力。

### 4.2 工作阶段

#### 1. 搜索阶段

- 大视场角搜索目标
- 红外图像生成和处理
- 目标初步探测

#### 2. 跟踪阶段

- 切换到小视场角
- 精确跟踪已发现目标
- 目标类型识别

#### 3. 锁定阶段

- 确认目标类型
- 持续精确跟踪
- 执行末段制导

## 4.3 关键技术

### 1. 红外图像生成

- 高分辨率图像采集
- 考虑目标温度、距离、大气衰减
- 背景噪声和干扰处理

### 2. 目标识别算法

- 信噪比计算和阈值判断
- 目标特征提取和匹配
- 识别概率评估

### 3. 多模式工作

- Search模式：大视场搜索
- Track模式：小视场跟踪
- Lock模式：锁定制导

### 4. 比例导引控制

- 目标状态估计
- 运动预测
- 制导加速度计算

## 4.4 技术特点

- 完全自主：无需外部制导信号
- 目标识别：具备目标类型识别能力
- 抗干扰：对烟幕、红外干扰有一定抗性
- 末段精确：末段制导精度高

# 5. 毫米波末制导导弹

## 5.1 工作原理

毫米波末制导导弹使用毫米波雷达进行目标探测和跟踪，具备全天候作战能力和强抗干扰性。

## 5.2 工作阶段

### 1. 雷达开机阶段

- 毫米波雷达系统启动
- 天线开始扫描
- 建立探测基准

### 2. 目标搜索阶段

- 大范围扫描搜索目标
- 雷达回波信号处理
- 目标初步定位

### 3. 目标跟踪阶段

- 锁定目标进行跟踪
- 连续测量目标位置
- 计算目标运动参数

### 4. 末制导阶段

- 精确跟踪目标
- 实时轨迹修正
- 引导导弹命中目标

## 5.3 关键技术

### 1. 毫米波雷达系统

- 工作在毫米波频段
- 具备一定的探测距离
- 高精度角度分辨率

## 2. 信号处理

- 雷达回波信号处理
- 目标检测算法
- 杂波抑制技术

## 3. 跟踪算法

- 目标航迹建立
- 运动状态估计
- 预测滤波

## 5.4 技术特点

- 全天候作战：不受天气条件影响
- 强抗干扰：对光电干扰免疫
- 穿透能力强：可穿透烟幕、雾霾
- 探测距离远：毫米波传播特性好

## 6. 复合制导导弹

### 6.1 工作原理

复合制导导弹集成多种制导方式，可以串行或并行工作，提供高可靠性和多重制导保障。

### 6.2 工作模式

#### 1. 串行模式

- 制导系统按优先级顺序激活
- 前一个系统失效后切换到下一个
- 提供制导冗余保障

#### 2. 并行模式

- 多个制导系统同时工作
- 根据融合策略选择最优制导信号
- 提高制导精度和可靠性

### 6.3 制导组件管理

#### 1. 激活触发器

- 发射时激活
- 飞行时间触发
- 距离触发
- 前级完成触发

#### 2. 制导切换逻辑

- 最大获取制导时间限制
- 最小稳定制导时间要求
- 失效后继续链条选项

### 6.4 典型配置

#### 1. 激光+红外复合制导

- 第一阶段：激光半主动制导
- 第二阶段：红外成像末制导
- 提供远程精确+末段自主能力

#### 2. 毫米波+红外复合制导

- 第一阶段：毫米波末制导
- 第二阶段：红外成像末制导



- 提供全天候+高精度能力

## 6.5 技术特点

- **高可靠性**：多重制导保障
- **适应性强**：适应不同作战环境
- **精度高**：多系统融合提高精度
- **复杂度高**：系统复杂，成本较高

## 7. 末敏弹

### 7.1 工作原理

末敏弹是一种智能子母弹，母弹在目标上空抛撒子弹，子弹通过螺旋扫描方式搜索和攻击地面目标。

### 7.2 工作阶段

#### 1. 母弹抛撒阶段

- 母弹飞抵目标上空后，时间引信作用
- 启动抛射装置，将末敏子弹按一定距离抛撒出来

#### 2. 子弹减速阶段

- 减速减旋装置动作，对子弹起减速、减旋、定向、稳向作用
- 启动热电池，达到规定值时开始对内部电子系统供电

#### 3. 第一期测距阶段

- 子弹以大着角下落
- 在中央控制器控制下，测距雷达开始第1期测距
- 测定子弹到地面的距离
- 达到预定高度时，抛去减速减旋装置
- 稳定扫描装置动作，带动子弹旋转

#### 4. 第二期测距阶段

- 稳定扫描装置带动末敏子弹稳态降落
- 在中央控制器控制下，测距雷达进行第2期测距
- 中央控制器完成对目标探测数据采集的准备工作
- 末敏子弹进入稳态扫描

#### 5. 目标探测阶段

- 末敏子弹进入威力有效高度
- 敏感探测器在中央控制器指令下进行工作扫描
- 在中央控制器控制下安装置解除最后一道保险
- 采用相邻2次扫描判定目标：
  - 第1次扫过目标后，向中央控制器报告目标信息
  - 第2次扫过目标后，将目标敏感数据与特征值比较

#### 6. 攻击/自毁阶段

- 如果第2次扫描确认是目标，且目标在威力窗口内：
  - 中央控制器下达指令起爆战斗部
  - 抛射出爆炸成形弹丸
  - 高速弹丸射向目标
  - 在目标来不及运动的瞬间命中并摧毁目标
- 如果第2次扫描判定为非目标：
  - 可以改换对象，继续探测其他潜在目标
- 如果一直没有发现目标：
  - 末敏子弹将在距离地面一定高度时自毁

### 7.3 关键特性

#### 1. 扫描特性

- 抛出的末敏子弹在实施扫描时相距一定距离
- 各自的扫描区相互衔接
- 避免击中同一目标或漏掉目标

2. 安全特性

- 多重保险装置
- 高度自毁保护
- 稳定扫描控制

3. 智能特性

- 自动目标识别
- 双次扫描确认
- 智能目标选择

7.4 技术特点

- 面杀伤能力：一发母弹可攻击多个目标
- 智能识别：具备目标自动识别能力
- 高杀伤效率：高速弹丸杀伤力强
- 适合集群目标：特别适合攻击装甲集群

8. 技术对比总结

制导类型	制导精度	抗干扰性	自主性	全天候	复杂度	适用场景
激光半主动	极高	中等	低	良好	中等	精确打击
激光驾束	极高	中等	低	良好	中等	精确制导
红外指令	高	中等	低	中等	中等	远程制导
红外成像	高	中等	高	中等	高	末段自主
毫米波	高	强	高	优秀	高	全天候作战
复合制导	极高	强	高	优秀	很高	高价值目标
末敏弹	中等	中等	高	良好	高	面目标攻击

9. 应用建议

1. 精确点目标：推荐激光半主动或激光驾束制导
2. 高价值目标：推荐复合制导系统
3. 恶劣天气：推荐毫米波制导
4. 集群目标：推荐末敏弹
5. 远程打击：推荐红外指令制导
6. 自主作战：推荐红外成像制导

# 使用说明

版本: 1.1.22

## 安装和使用

### C#/.NET 版本

#### 1. 下载和部署

1. 下载 ThreatSourceLibrary 包并解压
2. 按照以下标准结构部署文件:

```
MyApplication/
├── MyApp.exe           # 您的应用程序
├── lib/                # DLL库目录
│   ├── ThreatSource.dll # 主要的库文件
│   ├── ThreatSource.deps.json # 依赖配置文件
│   ├── ThreatSource.xml  # API 文档文件
│   └── AirTransmission.dll # 依赖库
└── data/              # 配置数据目录
    ├── missiles/      # 导弹配置文件
    │   ├── laser_semi_active/
    │   ├── laser_beam_rider/
    │   ├── ir_command/
    │   ├── ir_imaging/
    │   ├── mmw/
    │   ├── terminal_sensitive/
    │   └── composite/
    ├── indicators/    # 指示器配置文件
    │   ├── laser_designators/
    │   ├── laser_beam_riders/
    │   └── ir_trackers/
    ├── equipments/    # 目标设备配置文件
    ├── jammers/       # 干扰机配置文件
    ├── weathers/       # 天气配置文件
    └── warners/       # 告警器配置文件
```

3. 在 Visual Studio 中添加引用:
  - 右键项目 -> 添加 -> 引用
  - 浏览 -> 选择 lib/ThreatSource.dll

#### 2. 数据配置管理

威胁源库使用智能路径解析系统, 会自动在DLL上级目录查找 `data` 文件夹。您也可以手动指定数据路径:

```
// 使用自动路径解析 (推荐)
var dataManager = new ThreatSourceDataManager();

// 或手动指定数据路径
var dataManager = new ThreatSourceDataManager("C:/MyApp/configs");
```

### 3. 基本使用示例

```
using ThreatSource.Simulation;
using ThreatSource.Data;
using ThreatSource.Equipment;
using ThreatSource.Utills;

class Program
{
    static void Main()
    {
        try
        {
            // 1. 创建仿真管理器
            var simulationManager = new SimulationManager();

            // 2. 创建数据管理器和工厂
            var dataManager = new ThreatSourceDataManager();
            var factory = new ThreatSourceFactory(dataManager, simulationManager);

            // 3. 创建目标
            var targetInitialState = new KinematicState
            {
                Position = new Vector3D(1000, 0, 100),
                Orientation = new Orientation(0, 0, 0),
                Speed = 0
            };

            var targetProperties = new EquipmentProperties
            {
                InfraredRadiationIntensity = 100.0,
                MillimeterWaveRadiationIntensity = 50.0
            };

            var target = new Tank("target1", targetProperties, targetInitialState, simulation
Manager);

            // 4. 创建导弹 (使用工厂模式)
            var launchParams = new KinematicState
            {
                Position = new Vector3D(0, 0, 0),
                Orientation = new Orientation(0, 0, 0),
                Speed = 100
            };

            // 使用配置文件中的导弹型号
            var missile = factory.CreateMissile("missile1", "lsgm_001", "target1", launchPara
ms);

            // 5. 注册实体到仿真管理器
            simulationManager.RegisterEntity(target.Id, target);
            simulationManager.RegisterEntity(missile.Id, missile);

            // 6. 激活实体
            target.Activate();
            missile.Activate();

            // 7. 发射导弹
            missile.Fire();

            // 8. 启动仿真
            simulationManager.StartSimulation(0.01); // 10ms时间步长

            // 9. 仿真主循环
            double totalTime = 0;
            double maxTime = 60; // 最大仿真时间60秒
```

```

        while (missile.IsActive && totalTime < maxTime)
        {
            simulationManager.Update(0.01);
            totalTime += 0.01;

            // 可选: 输出状态信息
            if ((int)(totalTime * 100) % 100 == 0) // 每秒输出一次
            {
                Console.WriteLine($"时间: {totalTime:F1}s, 导弹位置: {missile.KState.Position}");
            }
        }

        // 10. 停止仿真
        simulationManager.StopSimulation();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"错误: {ex.Message}");
    }
}
}

```

## C++版本

C++项目有两种使用方式:

### 方式一: 动态加载 (推荐)

这种方式适合纯C++项目, 不需要配置CLR支持。

1. 下载 ThreatSourceNative 包并解压
2. 将 bin 目录下的所有 DLL 文件复制到程序目录
3. 将 include/threat\_source.h 添加到项目中
4. 按照标准结构部署数据文件

示例代码:

```

#include "threat_source.h"
#include <stdio.h>

int main() {
    // 初始化仿真
    if (TS_CreateSimulation() != THREATSOURCE_SUCCESS) {
        printf("Failed to create simulation\n");
        return 1;
    }

    // 创建导弹（使用配置文件）
    const char* missile_id = "missile1";
    const char* missile_model = "lsgm_001"; // 激光半主动制导导弹
    const char* target_id = "target1";

    int result = TS_CreateMissileFromConfig(
        missile_id,
        missile_model,
        target_id,
        0, 0, 0, // 初始位置
        0, 0, 0 // 初始朝向
    );

    if (result != THREATSOURCE_SUCCESS) {
        char error[256];
        TS_GetLastError(error, sizeof(error));
        printf("Failed to create missile: %s\n", error);
        return 1;
    }

    // 激活并发射导弹
    TS_ActivateMissile(missile_id);
    TS_FireMissile(missile_id);

    // 仿真主循环
    double deltaTime = 0.01;
    int is_active = 1;

    while (is_active) {
        TS_UpdateSimulation(deltaTime);
        TS_IsMissileActive(missile_id, &is_active);
    }

    // 清理
    TS_DestroySimulation();
    return 0;
}

```

## 方式二：CLR 集成

如果需要使用 .NET 的完整功能，可以选择 CLR 集成方式：

1. 配置项目属性：
  - C/C++ -> 常规 -> 公共语言运行时支持：/clr
  - 常规 -> 平台工具集：Visual Studio 2022 (v143)
  - 常规 -> .NET目标框架：net8.0
2. 引用 ThreatSource.dll

示例代码：

```

using namespace System;
using namespace ThreatSource::Simulation;
using namespace ThreatSource::Data;

int main() {
    try {
        // 创建仿真管理器和数据管理器
        auto manager = gcnew SimulationManager();
        auto dataManager = gcnew ThreatSourceDataManager();
        auto factory = gcnew ThreatSourceFactory(dataManager, manager);

        // 创建发射参数
        auto launchParams = gcnew KinematicState();
        launchParams->Position = Vector3D(0, 0, 0);
        launchParams->Orientation = Orientation(0, 0, 0);
        launchParams->Speed = 100;

        // 使用工厂创建导弹
        auto missile = factory->CreateMissile("missile1", "lsgm_001", "target1", launchParams
    );

        manager->RegisterEntity(missile->Id, missile);

        // 激活并发射导弹
        missile->Activate();
        missile->Fire();

        // 仿真主循环
        double deltaTime = 0.01;
        while (missile->IsActive) {
            manager->Update(deltaTime);
        }

        return 0;
    }
    catch (Exception^ e) {
        Console::WriteLine("错误: {0}", e->Message);
        return 1;
    }
}

```

## 支持的导弹类型

威胁源库支持以下导弹类型：

### 1. 激光半主动制导导弹 (LaserSemiActiveGuidance)

- 工作原理：利用目标反射的激光能量进行制导
- 配置文件：data/missiles/laser\_semi\_active/
- 制导特点：需要激光目标指示器持续照射目标

### 2. 激光驾束制导导弹 (LaserBeamRiderGuidance)

- 工作原理：沿着激光束路径飞行
- 配置文件：data/missiles/laser\_beam\_rider/
- 制导特点：跟踪激光束中心线，使用PID控制

### 3. 红外指令制导导弹 (InfraredCommandGuidance)

- 工作原理：通过红外信号接收指令进行制导
- 配置文件：data/missiles/ir\_command/
- 制导特点：导弹发射红外信号，地面测角仪跟踪并发送指令

### 4. 红外成像末制导导弹 (InfraredImagingTerminalGuidance)

- 工作原理：末段使用红外成像进行自主制导
- 配置文件：data/missiles/ir\_imaging/
- 制导特点：自主目标识别和跟踪

## 5. 毫米波末制导导弹 (MillimeterWaveTerminalGuidance)

- 工作原理：末段使用毫米波雷达进行自主制导
- 配置文件：data/missiles/mmw/
- 制导特点：全天候作战能力，抗干扰性强

## 6. 末敏弹 (TerminalSensitiveMissile)

- 工作原理：母弹释放子弹，子弹螺旋扫描攻击目标
- 配置文件：data/missiles/terminal\_sensitive/
- 制导特点：多传感器融合，智能目标识别

## 7. 复合制导导弹 (CompositeGuidance)

- 工作原理：集成多种制导方式，可串行或并行工作
- 配置文件：data/missiles/composite/
- 制导特点：高可靠性，多重制导保障

# 配置文件格式

威胁源库使用TOML格式的配置文件。以下是激光半主动制导导弹的配置示例：

```
# 激光半主动制导导弹配置示例
Type = "LaserSemiActiveGuidance"

[Name]
zh = "激光半主动制导导弹-001"
en = "Laser Semi-Active Guidance Missile-001"

[Properties]
Type = "LaserSemiActiveGuidance"
MaxSpeed = 800.0 # 最大速度 (m/s)
MaxFlightTime = 60.0 # 最大飞行时间 (秒)
MaxFlightDistance = 4000.0 # 最大飞行距离 (米)
MaxAcceleration = 50.0 # 最大加速度 (m/s²)
ProportionalNavigationCoefficient = 3.0 # 比例导引系数
LaunchAcceleration = 100.0 # 发射加速度 (m/s²)
MaxEngineBurnTime = 0.1 # 发动机燃烧时间 (秒)
CruiseTime = 5.0 # 巡航时间 (秒)
Mass = 22.0 # 质量 (kg)
ExplosionRadius = 5.0 # 爆炸半径 (米)
HitProbability = 0.9 # 命中概率
SelfDestructHeight = 0.0 # 自毁高度 (米)

[LaserSemiActiveGuidanceConfig]
MaxDetectionRange = 5000.0 # 最大探测距离 (米)
FieldOfViewAngle = 30.0 # 视场角 (度)
LockThreshold = 1.0e-12 # 锁定阈值 (瓦特)
ReflectionCoefficient = 0.2 # 反射系数
TargetReflectiveArea = 1.0 # 目标反射面积 (平方米)
```

# 基本用法

## 初始化仿真系统



```
// 创建仿真管理器
var simulationManager = new SimulationManager();

// 创建数据管理器（自动路径解析）
var dataManager = new ThreatSourceDataManager();

// 创建工厂
var factory = new ThreatSourceFactory(dataManager, simulationManager);
```

## 创建和配置实体

```
// 创建目标
var targetState = new KinematicState
{
    Position = new Vector3D(1000, 0, 100),
    Orientation = new Orientation(0, 0, 0),
    Speed = 0
};

var targetProperties = new EquipmentProperties
{
    InfraredRadiationIntensity = 100.0,
    MillimeterWaveRadiationIntensity = 50.0
};

var target = new Tank("target1", targetProperties, targetState, simulationManager);

// 创建导弹（从配置文件）
var launchParams = new KinematicState
{
    Position = new Vector3D(0, 0, 0),
    Orientation = new Orientation(0, 0, 0),
    Speed = 100
};

var missile = factory.CreateMissile("missile1", "lsgm_001", "target1", launchParams);
```

## 运行仿真

```
// 注册实体
simulationManager.RegisterEntity(target.Id, target);
simulationManager.RegisterEntity(missile.Id, missile);

// 激活实体
target.Activate();
missile.Activate();

// 发射导弹
missile.Fire();

// 启动仿真
simulationManager.StartSimulation(0.01);

// 仿真循环
while (missile.IsActive)
{
    simulationManager.Update(0.01);
}

// 停止仿真
simulationManager.StopSimulation();
```

## 事件处理

```
// 订阅导弹命中事件
simulationManager.SubscribeToEvent<TargetHitEvent>(evt =>
{
    Console.WriteLine($"目标 {evt.TargetId} 被导弹 {evt.SenderId} 命中");
});

// 订阅导弹爆炸事件
simulationManager.SubscribeToEvent<MissileExplodeEvent>(evt =>
{
    Console.WriteLine($"导弹在位置 {evt.Position} 爆炸");
});
```

## 数据管理

### 获取可用配置

```
var dataManager = new ThreatSourceDataManager();

// 获取所有可用的导弹型号
var availableMissiles = dataManager.GetAvailableMissiles();
foreach (var missileId in availableMissiles)
{
    Console.WriteLine($"可用导弹: {missileId}");
}

// 获取特定导弹的配置
var missileData = dataManager.GetMissile("lsgm_001");
Console.WriteLine($"导弹类型: {missileData.Type}");
Console.WriteLine($"最大速度: {missileData.Properties.MaxSpeed} m/s");
```

### 自定义数据路径

```
// 指定自定义数据路径
var customDataManager = new ThreatSourceDataManager("/path/to/custom/data");

// 或在运行时切换
var dataManager = new ThreatSourceDataManager();
// 数据会自动从 DLL上级目录/data 加载
```

## 部署要求

### 最低系统要求

- .NET 8.0 或更高版本
- Windows 10 或更高版本（推荐）
- 至少 100MB 可用磁盘空间

### 依赖项

- Tomlyn (TOML解析库)
- AirTransmission (空气传播计算库)

### 部署检查清单

1. 确保DLL文件在正确位置
2. 确保数据文件按标准结构组织
3. 确保配置文件格式正确 (TOML)
4. 确保应用程序有读取数据目录的权限

## 更多示例

更多详细示例和高级用法请参考：

- [仿真示例](#)
- [集成示例](#)

## 故障排除

### 常见问题

1. 数据文件未找到
  - 检查数据目录结构是否正确
  - 确认DLL和数据目录的相对位置
  - 查看调试输出中的路径解析信息
2. 配置文件解析错误
  - 检查TOML文件语法
  - 确认必需字段是否存在
  - 查看错误日志获取详细信息
3. 导弹创建失败
  - 确认导弹型号在配置文件中存在
  - 检查配置文件的完整性
  - 验证制导系统配置是否正确

# 威胁源仿真系统技术说明书

版本：1.1.22

## 1. 概述

### 1.1 文档目的

本文档旨在详细描述威胁源仿真系统的设计实现，包括：

1. 各型导弹的工作原理和关键算法
2. 系统组件的交互机制和接口定义
3. 数据管理系统和配置文件格式
4. 关键参数的配置和验证方法
5. 测试用例和验证流程

本文档可用于：

- 系统设计和实现的参考依据
- 测试验证的指导文档
- 系统维护和升级的技术支持
- 新功能开发的基础文档

### 1.2 适用范围

本文档适用于以下人员和场景：

1. 开发人员
  - 系统功能实现
  - 接口开发和维护
  - 算法优化和改进
  - 新功能扩展开发
2. 测试人员
  - 功能测试设计
  - 性能测试验证
  - 系统集成测试
  - 回归测试执行
3. 维护人员
  - 系统问题诊断
  - 参数配置调整
  - 性能监控分析
  - 系统升级维护
4. 使用场景
  - 导弹系统仿真
  - 制导算法验证
  - 性能指标评估
  - 战术效能分析

### 1.3 系统功能

1. 导弹类型支持

- 激光半主动制导导弹
- 激光驾束制导导弹
- 红外指令制导导弹
- 红外成像末制导导弹
- 毫米波末制导导弹
- 末敏弹
- 复合制导导弹（多模制导）

## 2. 核心功能

- 智能数据管理系统
- TOML配置文件支持
- 运动学仿真（基于KinematicState）
- 制导控制仿真
- 目标交互仿真
- 传感器仿真
- 事件系统
- 外部仿真集成

## 3. 数据管理功能

- 智能路径解析
- 多设备类型支持
- 配置文件热加载
- 数据验证和错误处理

## 4. 验证功能

- 单元测试验证
- 系统集成验证
- 性能指标验证
- 数据分析验证

## 5. 新增功能（1.1.22版本）

- SwerlingRCS回波模型
- 升力加速度计算系统
- 命中概率计算
- 运动状态随机噪声
- 统一状态信息管理（ElementStatusInfo）

# 1.4 参考文献

## 1. 导弹技术参考

- 《导弹制导与控制》，科学出版社
- 《现代制导技术》，国防工业出版社
- 《导弹动力学与控制》，航空工业出版社

## 2. 传感器技术参考

- 《红外成像技术》，科学出版社
- 《激光制导原理》，国防工业出版社
- 《毫米波雷达技术》，电子工业出版社

## 3. 仿真技术参考

- 《系统仿真理论与实践》，科学出版社
- 《飞行器建模与仿真》，航空工业出版社
- 《实时仿真系统》，国防工业出版社

## 4. 软件工程参考

- 《软件工程：实践者的研究方法》
- 《面向对象分析与设计》
- 《设计模式：可复用面向对象软件的基础》

# 2. 系统架构

## 2.1 总体架构

## 2.1.1 核心组件

### 1. 数据管理系统 (ThreatSourceDataManager)

- 智能路径解析：基于程序集位置自动查找数据目录
- TOML配置文件加载：支持多种设备类型的配置
- 数据缓存管理：内存中缓存已加载的配置数据
- 错误处理和日志：详细的加载过程跟踪和错误报告

### 2. 仿真管理系统 (SimulationManager)

- 仿真状态控制：Running/Paused/Stopped状态管理
- 时间同步控制：支持固定时间步长和外部时间同步
- 实体更新调度：线程安全的实体状态更新
- 天气系统集成：支持动态天气条件变化
- 外部仿真适配：支持与Unity、UE等外部引擎集成

### 3. 事件系统

- 发布订阅模式：类型安全的事件处理机制
- 事件分发处理：高效的事件路由和分发
- 事件队列管理：异步事件处理支持
- 外部事件集成：支持与外部系统的事件交换

### 4. 实体管理系统

- 实体注册与注销：线程安全的实体生命周期管理
- 实体状态维护：基于KinematicState的运动学状态管理
- 实体查询服务：高效的实体检索和类型过滤
- 碰撞检测：实体间的交互和碰撞处理

## 2.1.2 导弹组件架构

### 1. 基础导弹类 (BaseMissile)

- 继承自SimulationElement基类
- KinematicState运动学模型：位置、速度、朝向管理
- 飞行阶段管理：Launch/Cruise/TerminalGuidance/Explode/SelfDestruct
- 制导系统引用：统一的制导系统管理
- 升力加速度系统：LiftAcceleration计算和控制
- 事件处理机制：制导事件的接收和响应
- 自毁控制：基于时间、距离、高度的安全机制

### 2. 制导系统

- 激光半主动制导 (LaserSemiActiveGuidanceSystem)
- 激光驾束制导 (LaserBeamRiderGuidanceSystem)
- 红外指令制导 (InfraredCommandGuidanceSystem)
- 红外成像末制导 (InfraredImagingTerminalGuidanceSystem)
- 毫米波末制导 (MillimeterWaveTerminalGuidanceSystem)
- 末敏子弹制导 (TerminalSensitiveSubmunitionGuidanceSystem)
- 复合制导 (CompositeGuidanceSystem)

### 3. 传感器系统

- 激光探测器：四象限探测器、激光功率计算
- 红外探测器：热成像、目标识别、信噪比计算
- 毫米波雷达：目标探测、距离测量、抗干扰、SwerlingRCS模型
- 高度计：精确高度测量、地形跟踪
- 多传感器融合：数据融合算法、目标确认

## 2.1.3 目标组件架构

### 1. 基础目标类 (BaseTarget)

- 继承自SimulationElement基类
- 运动学模型：位置、速度、朝向的动态更新
- 状态管理：生命值、伤害系统、销毁机制
- 事件处理：目标辐射事件、命中事件的处理

### 2. 目标特征系统

- 红外辐射特征：温度分布模式、辐射强度计算
- 雷达散射特征：RCS模式、角度依赖性、SwerlingRCS模型

- 激光反射特征：反射系数、有效反射面积
- 毫米波特征：散射截面、频率响应

## 2.2 坐标系统

### 2.2.1 坐标定义

1. 世界坐标系（右手系）
  - X轴：指向东方
  - Y轴：指向上方
  - Z轴：指向北方
  - 原点：仿真场景中心
2. 导弹本体坐标系
  - X轴：指向导弹右翼
  - Y轴：指向导弹上方
  - Z轴：指向导弹后方（前向为-Z轴）
  - 原点：导弹质心

### 2.2.2 朝向系统

1. Orientation欧拉角定义
  - 偏航角（Yaw）：绕Y轴旋转，范围 $[-\pi, \pi]$
  - 俯仰角（Pitch）：绕X轴旋转，范围 $[-\pi/2, \pi/2]$
  - 滚转角（Roll）：绕Z轴旋转，范围 $[-\pi, \pi]$
2. 前向向量计算
  - ToVector()方法返回-Z轴方向的单位向量
  - 默认朝向(0,0,0)对应世界坐标系的(0,0,-1)
  - 使用右手坐标系和逆时针旋转约定

### 2.2.3 运动学状态管理

1. KinematicState类
  - Position：三维位置向量
  - Orientation：欧拉角朝向
  - Velocity：三维速度向量
  - Speed：标量速度（自动与Velocity同步）
2. 状态同步机制
  - Speed设置时自动更新Velocity方向
  - Velocity设置时自动更新Speed大小
  - Orientation与速度方向的智能关联

### 2.2.4 单位说明

1. 空间单位
  - 距离：米 (m)
  - 速度：米/秒 (m/s)
  - 加速度：米/秒<sup>2</sup> (m/s<sup>2</sup>)
  - 角度：弧度 (rad)
2. 时间单位
  - 仿真时间：秒 (s)
  - 事件时间戳：微秒 (us)
  - 更新周期：秒 (s)

## 2.3 数据管理架构

### 2.3.1 智能路径解析系统

## 1. 路径解析策略

系统采用两级路径解析策略：

- 优先级1：DLL上级目录下的data文件夹
  - 获取当前程序集的物理位置
  - 解析DLL所在目录
  - 获取上级目录
  - 组合形成数据路径
- 优先级2：相对路径回退
  - 当自动解析失败时使用相对路径"data"
  - 保持向后兼容性

## 2. 部署结构支持

标准部署结构如下：

- 应用程序根目录包含主程序
- lib子目录存放DLL库文件
- data子目录存放配置数据（自动发现）
- data目录下按设备类型分类存放TOML配置文件

### 2.3.2 TOML配置文件系统

#### 1. 配置文件结构

- 使用Tomlyn库进行TOML解析
- 支持嵌套配置结构
- 类型安全的配置映射
- 详细的错误报告和验证

#### 2. 设备类型支持

- 导弹配置：包含制导系统、飞行参数、武器参数
- 指示器配置：激光指示器、红外测角仪等
- 传感器配置：告警器、探测器等
- 设备配置：目标设备、平台等
- 天气配置：天气条件、环境参数
- 干扰机配置：各类干扰设备

#### 3. 复合制导配置

复合制导系统支持多个制导组件的配置：

- 每个组件包含名称、类型、激活触发器
- 支持基于时间、距离、事件的激活条件
- 可配置最大获取时间和稳定制导时间
- 支持失效后的继续链条选项

## 2.4 状态信息管理架构

### 2.4.1 ElementStatusInfo统一系统

#### 1. 统一状态信息类

ElementStatusInfo提供标准化的状态信息管理：

- Id：实体唯一标识符
- ElementType：实体类型名称
- KState：运动学状态信息
- IsActive：活动状态标志
- Timestamp：状态时间戳
- ExtendedProperties：扩展属性字典

#### 2. GetStatusInfo方法

所有仿真实体统一实现GetStatusInfo方法：

- 基类提供基础状态信息
- 子类添加特定扩展属性
- 支持嵌套状态信息（如制导系统状态）



- 自动格式化输出

### 3. 状态信息层次结构

- SimulationElement: 基础状态信息
- BaseMissile: 导弹特有状态 (飞行时间、阶段、制导状态)
- BaseGuidanceSystem: 制导系统状态 (制导加速度、干扰状态)
- BaseEquipment: 设备状态 (生命值、属性信息)
- BaseSensor: 传感器状态 (干扰状态、传感器数据)

## 2.5 升力系统架构

### 2.5.1 LiftModel升力模型

#### 1. 升力计算原理

基于俯仰角 (攻角) 计算升力加速度:

- 有效俯仰角范围:  $-5^{\circ}$ 到 $15^{\circ}$
- 升力平衡点:  $5^{\circ}$ 俯仰角
- 升力系数:  $1.0 \text{ m/s}^2/\text{度}$

#### 2. 升力加速度计算

升力加速度计算公式:

- $\text{combined\_vertical\_acceleration} = (\text{pitch} - 5^{\circ}) \times 1.0$
- $\text{lift\_acceleration} = \text{combined\_vertical\_acceleration} + g$
- 返回垂直方向的升力向量

## 2.6 仿真管理架构

### 2.6.1 仿真状态管理

#### 1. 状态定义

仿真系统定义三种基本状态:

- Stopped: 仿真停止状态
- Running: 仿真运行状态
- Paused: 仿真暂停状态

#### 2. 状态转换控制

- 启动仿真: 从停止状态转为运行状态, 需指定时间步长
- 暂停仿真: 从运行状态转为暂停状态
- 恢复仿真: 从暂停状态转为运行状态
- 停止仿真: 转为停止状态并重置仿真时间

### 2.6.2 时间同步机制

#### 1. 内部时间管理

- 固定时间步长: 支持恒定的仿真步长
- 累积时间跟踪: CurrentTime属性记录仿真时间
- 实体同步更新: 所有实体使用统一的时间步长

#### 2. 外部时间同步

- 支持外部驱动的时间同步
- 时间同步事件通知时间变化
- 适配器时间同步通知外部仿真系统时间变化

### 2.6.3 天气系统集成

#### 1. 天气模型

天气系统包含以下参数:

- 天气类型: 晴朗、多云、雨天等
- 温度: 摄氏度

- 湿度：百分比
- 气压：百帕
- 风速：米每秒
- 风向：度数
- 能见度：公里

## 2. 天气影响

- 激光传播：大气透过率影响
- 红外探测：温度和湿度影响
- 毫米波传播：降雨衰减效应
- 目标特征：环境温度对红外辐射的影响
- 导弹运动：风速对轨迹的影响

## 2.7 事件系统架构

### 2.7.1 事件类型体系

#### 1. 基础事件类

所有仿真事件继承自基础事件类，包含：

- 发送者ID：标识事件来源实体
- 时间戳：事件发生的时间

#### 2. 导弹相关事件

- 导弹发射事件：导弹发射时触发
- 导弹爆炸事件：导弹爆炸时触发
- 飞行阶段变化事件：导弹阶段转换时触发
- 制导状态变化事件：制导状态改变时触发

#### 3. 制导相关事件

- 激光照射事件：激光指示器照射目标
- 激光波束事件：激光驾束仪发射波束
- 红外制导指令事件：红外测角仪发送制导指令
- 导弹红外信号事件：导弹发射红外信号

#### 4. 目标相关事件

- 目标命中事件：导弹命中目标
- 目标摧毁事件：目标被摧毁
- 目标辐射特征事件：目标发射辐射信号

#### 5. 系统事件

- 实体销毁事件：仿真实体被销毁
- 天气变化事件：天气条件发生变化
- 时间同步事件：仿真时间同步

### 2.7.2 事件处理机制

#### 1. 发布订阅模式

事件系统采用类型安全的发布订阅模式：

- 事件订阅：通过泛型方法订阅特定类型事件
- 事件发布：发布事件到所有订阅者
- 自动类型匹配：基于事件类型自动路由到对应处理器

#### 2. 类型安全处理

- 泛型事件处理器：确保类型安全
- 自动类型分发：基于事件类型的自动路由
- 异常隔离：单个处理器异常不影响其他处理器

#### 3. 外部系统集成

- 双向事件流：支持向外部系统发布和接收事件
- 适配器模式：通过ISimulationAdapter接口集成
- 事件转换：自动处理内外部事件格式转换

## 2.8 实体管理架构

## 2.8.1 实体基类设计

### 1. SimulationElement基类

所有仿真实体继承自SimulationElement基类，提供：

- 唯一标识符：实体ID
- 运动学状态：KinematicState对象
- 活动状态：IsActive标志
- 仿真管理器引用：用于事件发布和实体查询
- 抽象更新方法：子类实现具体更新逻辑
- 生命周期方法：激活和停用方法

### 2. KinematicState运动学状态

运动学状态包含：

- 位置：三维空间坐标
- 速度：三维速度向量
- 朝向：三维朝向向量
- 角速度：三维角速度向量

## 2.8.2 实体生命周期管理

### 1. 注册机制

- 线程安全的实体注册
- 重复ID检查：防止ID冲突
- 类型验证：确保实体类型正确

### 2. 查询机制

- 按ID查询实体：根据唯一标识符查找
- 按类型查询实体：获取特定类型的所有实体
- 获取所有实体：返回系统中的所有实体

### 3. 注销机制

- 安全移除实体：从系统中移除指定实体
- 级联清理：清理相关的事件订阅和引用
- 资源释放：确保内存和资源正确释放

## 2.8.3 线程安全机制

### 1. 锁策略

实体管理使用锁机制确保线程安全：

- 注册操作使用互斥锁保护
- 查询操作使用只读锁优化性能
- 注销操作使用写锁确保一致性

### 2. 并发访问

- 读写分离：查询操作使用只读集合
- 原子操作：关键状态变更的原子性保证
- 死锁避免：合理的锁顺序和超时机制

## 2.9 数据流架构

### 2.9.1 配置数据流

配置数据流程包括以下步骤：

1. TOML配置文件作为数据源
2. ThreatSourceDataManager负责数据管理
3. 智能路径解析定位配置文件
4. 文件加载和TOML解析
5. 数据验证确保配置正确性
6. 内存缓存提高访问效率
7. 实体创建使用缓存的配置数据

## 2.9.2 仿真数据流

仿真数据流程包括：

1. 仿真管理器控制整体流程
2. 时间同步确保一致性
3. 实体更新处理状态变化
4. 事件处理响应系统事件
5. 状态同步维护数据一致性
6. 外部系统集成支持第三方平台

## 2.9.3 事件数据流

事件数据流程包括：

1. 事件发布者产生事件
2. 事件系统接收和管理事件
3. 类型分发根据事件类型路由
4. 处理器调用执行具体处理逻辑
5. 外部适配器处理外部系统集成
6. 外部系统接收和响应事件

# 2.10 接口定义

## 2.10.1 核心接口

### 1. ISimulationManager接口

- 仿真控制：启动、暂停、恢复、停止仿真
- 事件管理：发布、订阅、取消订阅事件
- 实体管理：注册、注销、查询实体
- 外部集成：设置适配器、处理外部事件

### 2. ISimulationAdapter接口

- 实体查询：获取实体和实体状态
- 事件交换：向外部发布事件和接收外部事件
- 时间同步：同步仿真时间
- 环境设置：设置实体环境数据

### 3. ISimulationElement接口

- 状态管理：更新状态、获取状态信息
- 生命周期：激活和停用实体
- 属性访问：ID、运动学状态、活动状态

## 2.10.2 数据接口

### 1. 配置数据接口

- 获取导弹配置：根据型号获取导弹配置数据
- 获取指示器配置：根据型号获取指示器配置数据
- 获取传感器配置：根据型号获取传感器配置数据
- 获取设备配置：根据型号获取设备配置数据

### 2. 查询接口

- 获取可用导弹列表：返回所有可用的导弹型号
- 获取可用指示器列表：返回所有可用的指示器型号
- 获取可用传感器列表：返回所有可用的传感器型号

# 3. 数据管理系统

## 3.1 ThreatSourceDataManager

### 3.1.1 核心功能

威胁源数据管理器是系统的核心数据管理组件，负责：

### 1. 智能路径解析

- 基于程序集位置自动查找数据目录
- 支持DLL上级目录的data文件夹自动发现
- 提供相对路径回退机制确保兼容性
- 详细的路径解析日志和错误报告

### 2. 配置文件加载

- 支持TOML格式配置文件
- 递归搜索子目录中的配置文件
- 按文件名自动识别设备型号
- 类型安全的配置数据映射

### 3. 数据缓存管理

- 内存中缓存已加载的配置数据
- 按设备类型分类存储
- 高效的数据检索和访问
- 支持配置数据的热更新

### 4. 错误处理和验证

- 详细的加载过程跟踪
- 配置文件格式验证
- 缺失字段检查和默认值处理
- 完整的错误日志和异常处理

## 3.1.2 支持的设备类型

### 1. 导弹配置 (MissileData)

- 基本属性：速度、加速度、飞行时间等
- 制导系统配置：各种制导方式的参数
- 武器参数：爆炸半径、命中概率等
- 复合制导：多制导系统的组合配置
- 升力参数：巡航攻角、制导下视角等

### 2. 指示器配置 (IndicatorData)

- 激光指示器：功率、波长、发散角等
- 激光驾束仪：波束参数、控制场直径等
- 红外测角仪：跟踪距离、视场角等

### 3. 传感器配置 (SensorData)

- 告警器：探测阈值、响应时间等
- 探测器：灵敏度、工作频段等
- 雷达：功率、频率、天线参数等

### 4. 设备配置 (EquipmentData)

- 目标设备：RCS模式、热特征等
- 平台设备：运动参数、载荷能力等
- 环境设备：地形、建筑物等

### 5. 天气配置 (WeatherData)

- 天气类型：晴朗、多云、雨天等
- 环境参数：温度、湿度、风速等
- 大气条件：能见度、气压等

### 6. 干扰机配置 (JammerData)

- 激光干扰机：干扰功率、波长等
- 红外干扰机：干扰强度、频谱等
- 毫米波干扰机：干扰带宽、功率等

## 3.1.3 智能路径解析实现

### 1. GetDataPath()方法

智能路径解析的核心实现：

```
优先级1: DLL上级目录/data
- 获取Assembly.GetExecutingAssembly().Location
- 解析DLL所在目录的父目录
- 组合形成data路径并验证存在性

优先级2: 相对路径回退
- 使用FALLBACK_DATA_PATH = "data"
- 保持向后兼容性
```

## 2. 路径解析日志

详细的调试信息输出：

- DLL位置信息
- 解析的data目录路径
- 目录存在性检查结果
- 回退路径使用情况

### 3.1.4 配置文件结构

#### 1. 基本结构

- Type字段：定义设备类型
- Name字段：多语言名称支持
- Properties字段：设备属性配置
- 专用配置字段：特定制导系统的配置

#### 2. 复合制导配置

- GuidanceSuite数组：多个制导组件
- 每个组件包含：名称、类型、激活条件
- 支持串行和并行制导模式
- 可配置制导切换逻辑

#### 3. 配置验证

- 必需字段检查
- 数值范围验证
- 类型匹配验证
- 依赖关系检查

## 3.2 配置文件格式

### 3.2.1 TOML格式优势

#### 1. 可读性强

- 类似INI文件的简洁语法
- 支持注释和文档化
- 层次化的配置结构
- 易于人工编辑和维护

#### 2. 类型安全

- 强类型数据支持
- 自动类型转换
- 数组和表格支持
- 嵌套结构支持

#### 3. 工具支持

- 使用Tomlyn库进行解析
- 完整的错误报告
- 性能优化的解析器
- .NET生态系统集成

### 3.2.2 配置文件组织

#### 1. 目录结构

- 按设备类型分目录存放

- 支持子目录进一步分类
- 文件名即为设备型号
- 统一的.toml文件扩展名

## 2. 命名规范

- 设备型号作为文件名
- 使用下划线分隔符
- 版本号后缀支持
- 描述性的目录名称

## 3. 版本管理

- 配置文件版本控制
- 向后兼容性保证
- 升级路径规划
- 变更日志维护

# 3.3 数据加载流程

## 3.3.1 初始化流程

### 1. 路径解析

- 获取程序集位置
- 计算数据目录路径
- 验证目录存在性
- 记录解析结果

### 2. 目录扫描

- 递归扫描配置目录
- 识别TOML配置文件
- 按设备类型分类
- 构建文件清单

### 3. 文件加载

- 逐个加载配置文件
- TOML格式解析
- 数据类型转换
- 错误处理和记录

### 4. 数据验证

- 配置完整性检查
- 数值范围验证
- 依赖关系验证
- 默认值填充

## 3.3.2 缓存管理

### 1. 内存缓存

- 按设备类型分类存储
- 使用字典结构快速查找
- 支持并发访问
- 内存使用优化

### 2. 缓存策略

- 一次性加载所有配置
- 延迟加载特定配置
- 缓存失效和更新
- 内存压力管理

### 3. 数据访问

- 类型安全的访问接口
- 异常处理和错误报告
- 性能监控和优化
- 并发访问控制

### 3.3.3 复合制导数据处理

#### 1. GuidanceSuite后处理

复合制导配置的特殊处理：

- 遍历GuidanceSuite中的每个组件
- 根据GuidanceSystemType匹配顶层配置
- 将SpecificConfig指向对应的配置对象
- 支持的制导类型映射：
  - "millimeterwave" → MillimeterWaveGuidanceConfig
  - "infraredimagingterminalguidance" → InfraredImagingGuidanceConfig
  - "lasersemiactiveguidance" → LaserSemiActiveGuidanceConfig
  - "laserbeamriderguidance" → LaserBeamRiderGuidanceConfig
  - "infraredcommandguidance" → InfraredCommandGuidanceConfig

#### 2. 配置验证和错误处理

- 检查SpecificConfig是否成功关联
- 验证顶层配置的存在性
- 记录详细的调试信息
- 处理配置缺失的情况

## 4. 导弹类型及特性

### 4.1 导弹基类架构

#### 4.1.1 BaseMissile基类

##### 1. 核心属性

- 继承自SimulationElement基类
- KinematicState运动学状态管理
- 飞行阶段状态机：Launch/Cruise/TerminalGuidance/Explode/SelfDestruct
- 制导系统引用：统一的制导系统管理接口
- 升力加速度系统：基于俯仰角的升力计算
- 事件处理机制：制导事件的接收和响应

##### 2. 飞行阶段管理

- Launch阶段：发射初期的弹道飞行
- Cruise阶段：中段巡航飞行
- TerminalGuidance阶段：末段制导飞行
- Explode阶段：爆炸状态
- SelfDestruct阶段：自毁状态

##### 3. 制导系统集成

- 制导系统引用管理
- 制导加速度计算和应用
- 制导事件的处理和响应
- 多制导系统的切换和管理

##### 4. 升力系统集成

- 基于俯仰角的升力计算
- 巡航攻角和制导下视角配置
- 升力加速度与制导加速度的合成
- 飞行阶段相关的升力参数调整

#### 4.1.2 运动学模型

##### 1. KinematicState状态管理

- Position：三维位置向量
- Orientation：欧拉角朝向（前向为-Z轴）
- Velocity：三维速度向量
- Speed：标量速度（与Velocity自动同步）

##### 2. 加速度合成



- GuidanceAcceleration: 制导系统产生的加速度
- ThrustAcceleration: 推力加速度
- LiftAcceleration: 升力加速度
- DragAcceleration: 空气阻力加速度
- GravityAcceleration: 重力加速度

### 3. 运动状态随机噪声

- 根据飞行阶段设置不同的噪声系数
- Launch阶段: 较大的初始噪声
- Cruise阶段: 中等的巡航噪声
- TerminalGuidance阶段: 较小的制导噪声

## 4.1.3 生命周期管理

### 1. 自毁控制机制

- 基于飞行时间的自毁
- 基于飞行距离的自毁
- 基于高度的安全自毁
- 基于制导失效的自毁

### 2. 命中检测

- 目标距离检测
- 爆炸半径计算
- 命中概率计算
- 目标伤害评估

### 3. 状态信息管理

- ElementStatusInfo统一状态信息
- 飞行时间、阶段、制导状态等扩展属性
- GetStatusInfo方法的统一实现
- 嵌套制导系统状态信息

## 4.2 激光半主动制导导弹

### 4.2.1 工作原理

激光半主动制导导弹通过接收目标反射的激光能量进行制导:

#### 1. 激光照射阶段

- 激光指示器持续照射目标
- 目标表面反射激光能量
- 导弹上的激光探测器接收反射信号
- 四象限探测器确定目标方位

#### 2. 制导控制阶段

- 计算目标相对于导弹轴线的角度偏差
- 根据偏差角度生成制导指令
- 比例导引法计算制导加速度
- 控制导弹飞向目标

#### 3. 抗干扰能力

- 激光功率阈值检测
- 信号质量评估
- 干扰识别和抑制
- 制导精度保持

### 4.2.2 关键算法

#### 1. 四象限探测器算法

- 四个象限的激光功率分布计算
- 基于距离平方反比定律的功率衰减
- 目标反射系数和有效反射面积
- 大气透过率对激光传播的影响

#### 2. 比例导引算法

- 视线角速度计算
- 导航比例系数设置
- 制导加速度矢量计算
- 制导指令的平滑处理

### 3. 干扰处理算法

- 激光干扰功率计算
- 信噪比评估
- 干扰状态判断
- 制导精度降级处理

## 4.2.3 技术特点

### 1. 制导精度

- 高精度的目标跟踪能力
- 亚米级的命中精度
- 全天候作战能力
- 抗电子干扰能力

### 2. 作战灵活性

- 发射后锁定能力
- 多目标攻击能力
- 复杂环境适应性
- 快速反应能力

### 3. 技术限制

- 需要持续激光照射
- 受天气条件影响
- 激光功率需求较高
- 易受激光干扰影响

## 4.3 激光驾束制导导弹

### 4.3.1 工作原理

激光驾束制导导弹通过跟踪激光波束进行制导：

#### 1. 波束跟踪阶段

- 激光驾束仪发射编码激光波束
- 导弹尾部的激光接收器接收波束信号
- 解码波束中的制导指令信息
- 计算导弹相对于波束轴线的偏差

#### 2. 制导控制阶段

- 根据波束偏差计算制导指令
- 控制导弹保持在波束中心
- 跟随波束指向目标飞行
- 实现精确的目标攻击

#### 3. 波束编码机制

- 时间编码的制导指令
- 空间编码的位置信息
- 频率编码的控制信号
- 抗干扰的编码算法

### 4.3.2 关键算法

#### 1. 波束跟踪算法

- 激光功率分布检测
- 波束中心位置计算
- 导弹偏差角度计算
- 制导指令生成

#### 2. 编码解码算法

- 时间序列信号处理
- 数字滤波和降噪
- 指令信息提取
- 错误检测和纠正

### 3. 控制场算法

- 控制场直径计算
- 有效制导区域确定
- 边界条件处理
- 制导精度评估

## 4.3.3 技术特点

### 1. 制导优势

- 高精度的波束跟踪
- 强抗干扰能力
- 全程制导控制
- 多目标攻击能力

### 2. 系统特点

- 需要专用激光驾束仪
- 波束功率要求较高
- 制导距离受限
- 对大气条件敏感

## 4.4 红外指令制导导弹

### 4.4.1 工作原理

红外指令制导导弹通过接收红外制导指令进行制导：

#### 1. 目标跟踪阶段

- 红外测角仪跟踪目标的红外辐射
- 计算目标的角度位置
- 预测目标的运动轨迹
- 生成拦截制导指令

#### 2. 指令传输阶段

- 红外测角仪发射编码红外信号
- 导弹接收红外制导指令
- 解码指令中的制导信息
- 执行制导机动动作

#### 3. 制导控制阶段

- 根据制导指令调整飞行方向
- 实现对目标的精确拦截
- 保持与测角仪的通信链路
- 完成目标攻击任务

### 4.4.2 关键算法

#### 1. 目标跟踪算法

- 红外图像处理
- 目标识别和分类
- 运动轨迹预测
- 拦截点计算

#### 2. 指令编码算法

- 制导指令数字化
- 红外信号调制
- 抗干扰编码
- 传输错误检测

#### 3. 制导控制算法

- 指令解码处理
- 制导加速度计算
- 飞行轨迹修正
- 目标拦截控制

### 4.4.3 技术特点

#### 1. 制导能力

- 远距离制导能力
- 多目标攻击能力
- 复杂环境适应性
- 高精度拦截能力

#### 2. 系统要求

- 需要红外测角仪支持
- 红外通信链路要求
- 目标红外特征依赖
- 大气透过率影响

## 4.5 红外成像末制导导弹

### 4.5.1 工作原理

红外成像末制导导弹在末段使用红外成像传感器进行自主制导：

#### 1. 目标搜索阶段

- 红外成像传感器扫描搜索区域
- 检测目标的红外辐射特征
- 识别和确认目标类型
- 锁定目标进入跟踪状态

#### 2. 目标跟踪阶段

- 持续跟踪目标的红外图像
- 计算目标的角度位置和距离
- 预测目标的运动状态
- 生成制导控制指令

#### 3. 精确制导阶段

- 根据目标位置计算制导加速度
- 控制导弹精确飞向目标
- 实现高精度的目标攻击
- 完成末段制导任务

### 4.5.2 关键算法

#### 1. 图像处理算法

- 红外图像增强
- 目标检测和分割
- 特征提取和匹配
- 目标识别和分类

#### 2. 目标跟踪算法

- 卡尔曼滤波跟踪
- 粒子滤波跟踪
- 多假设跟踪
- 目标状态估计

#### 3. 制导控制算法

- 比例导引法
- 最优制导法
- 预测制导法
- 自适应制导法

### 4.5.3 技术特点

#### 1. 制导优势

- 自主制导能力
- 高精度攻击能力
- 抗干扰能力强
- 全天候作战能力

#### 2. 技术挑战

- 图像处理复杂度高
- 计算资源需求大
- 目标特征依赖性
- 环境适应性要求

## 4.6 毫米波末制导导弹

### 4.6.1 工作原理

毫米波末制导导弹使用毫米波雷达进行末段制导：

#### 1. 目标探测阶段

- 毫米波雷达发射电磁波
- 接收目标反射的回波信号
- 计算目标的距离和方位
- 识别和确认目标类型

#### 2. 目标跟踪阶段

- 持续跟踪目标的雷达回波
- 测量目标的距离、方位和速度
- 预测目标的运动轨迹
- 生成制导控制指令

#### 3. 精确制导阶段

- 根据雷达测量数据计算制导加速度
- 控制导弹精确飞向目标
- 实现高精度的目标攻击
- 完成末段制导任务

### 4.6.2 关键算法

#### 1. 雷达信号处理算法

- 脉冲压缩处理
- 多普勒频移检测
- 距离和速度测量
- 信号检测和估计

#### 2. 目标检测算法

- 恒虚警率检测
- 目标航迹关联
- 多目标跟踪
- 目标识别分类

#### 3. SwerlingRCS模型

- Swerling I型：慢起伏目标
- Swerling II型：快起伏目标
- Swerling III型：慢起伏目标（改进）
- Swerling IV型：快起伏目标（改进）
- RCS统计特性建模
- 检测概率计算

#### 4. 制导控制算法

- 比例导引法
- 增强比例导引法
- 最优制导法
- 自适应制导法

### 4.6.3 技术特点

- 1. 制导优势
  - 全天候作战能力
  - 强穿透能力
  - 高精度测距能力
  - 抗光电干扰能力
- 2. 技术特点
  - 毫米波频段特性
  - 高分辨率成像能力
  - 多目标处理能力
  - 复杂环境适应性
- 3. 环境影响
  - 降雨衰减效应
  - 大气吸收影响
  - 地面杂波干扰
  - 多径传播效应

## 4.7 末敏弹

### 4.7.1 工作原理

末敏弹是一种特殊的智能弹药，具有自主目标搜索和攻击能力：

- 1. 抛撒阶段
  - 母弹在预定高度抛撒子弹
  - 子弹展开降落伞减速下降
  - 启动传感器系统进行目标搜索
  - 进入自主搜索攻击模式
- 2. 目标搜索阶段
  - 红外传感器扫描地面目标
  - 毫米波雷达探测金属目标
  - 多传感器融合确认目标类型
  - 选择最优攻击目标
- 3. 精确攻击阶段
  - 计算目标的精确位置
  - 控制子弹飞向目标
  - 在最佳位置引爆战斗部
  - 实现对目标的有效毁伤

### 4.7.2 关键算法

- 1. 目标搜索算法
  - 螺旋扫描模式
  - 目标检测算法
  - 目标分类识别
  - 威胁等级评估
- 2. 多传感器融合算法
  - 红外和毫米波数据融合
  - 目标特征匹配
  - 置信度计算
  - 决策融合算法
- 3. 攻击控制算法
  - 攻击时机选择
  - 攻击角度优化
  - 战斗部引爆控制
  - 毁伤效果评估

### 4.7.3 技术特点

#### 1. 智能化特点

- 自主目标识别
- 智能攻击决策
- 多目标处理能力
- 复杂环境适应性

#### 2. 作战优势

- 大面积搜索能力
- 高精度攻击能力
- 多目标同时攻击
- 强生存能力

## 4.8 复合制导导弹

### 4.8.1 工作原理

复合制导导弹集成多种制导方式，提供更强的作战能力：

#### 1. 多模制导系统

- 毫米波雷达制导
- 红外成像制导
- 激光半主动制导
- 惯性导航制导

#### 2. 制导模式切换

- 基于飞行阶段的自动切换
- 基于环境条件的智能切换
- 基于目标特性的优化切换
- 基于干扰情况的应急切换

#### 3. 数据融合处理

- 多传感器数据融合
- 制导信息综合处理
- 目标状态估计优化
- 制导精度提升

### 4.8.2 关键算法

#### 1. 制导模式选择算法

- 环境条件评估
- 目标特性分析
- 制导性能预测
- 最优模式选择

#### 2. 数据融合算法

- 卡尔曼滤波融合
- 贝叶斯融合
- 证据理论融合
- 神经网络融合

#### 3. 制导切换算法

- 切换条件判断
- 切换时机选择
- 状态平滑过渡
- 制导连续性保证

### 4.8.3 技术特点

#### 1. 制导优势

- 多重制导保险
- 强抗干扰能力
- 高制导精度

- 复杂环境适应性
- ## 2. 系统复杂性

- 多传感器集成
- 复杂控制逻辑
- 高计算资源需求
- 系统可靠性要求

## 4.9 命中概率计算

### 4.9.1 命中概率模型

1. 基础概率模型
  - 基于导弹类型的基础命中概率
  - 基于目标距离的概率修正
  - 基于制导精度的概率计算
  - 基于环境条件的概率调整
2. 距离影响因子
  - 近距离高命中概率
  - 远距离概率衰减
  - 最优攻击距离
  - 有效攻击范围
3. 制导精度影响
  - 制导系统精度等级
  - 传感器性能影响
  - 干扰条件影响
  - 目标特性影响

### 4.9.2 概率计算算法

1. 综合概率计算
  - 基础概率 × 距离因子 × 制导因子 × 环境因子
  - 各因子的权重分配
  - 概率范围限制
  - 随机性引入
2. 动态概率更新
  - 实时距离测量
  - 制导状态监控
  - 环境条件变化
  - 概率实时更新
3. 命中判定算法
  - 概率阈值设定
  - 随机数生成
  - 命中结果判定
  - 结果统计分析

## 5. 验证方法

### 5.1 单元测试验证

#### 5.1.1 数据管理系统测试

1. ThreatSourceDataManager测试
  - 智能路径解析功能验证
  - TOML配置文件加载测试
  - 数据缓存机制验证
  - 错误处理和异常测试
  - 复合制导配置后处理验证



## 2. 配置文件格式验证

- TOML语法正确性检查
- 必需字段完整性验证
- 数据类型匹配验证
- 数值范围合理性检查
- 依赖关系一致性验证

## 3. 路径解析测试

- DLL位置获取测试
- 上级目录data文件夹查找
- 回退路径机制验证
- 不同部署结构适应性测试
- 路径解析日志验证

## 5.1.2 运动学系统测试

### 1. KinematicState测试

- Position、Orientation、Velocity状态管理
- Speed与Velocity的自动同步
- 坐标系转换正确性验证
- 前向方向(-Z轴)计算验证
- 状态更新和同步机制测试

### 2. 升力系统测试

- LiftModel升力计算验证
- 俯仰角范围(-5°到15°)测试
- 升力平衡点(5°)验证
- 升力系数(1.0 m/s<sup>2</sup>/度)测试
- 垂直加速度合成验证

### 3. 加速度合成测试

- 制导加速度计算验证
- 推力加速度应用测试
- 升力加速度集成验证
- 阻力加速度计算测试
- 重力加速度合成验证

## 5.1.3 制导系统测试

### 1. 激光半主动制导测试

- 四象限探测器算法验证
- 激光功率分布计算测试
- 目标反射系数应用验证
- 比例导引算法测试
- 干扰处理算法验证

### 2. 激光驾束制导测试

- 波束跟踪算法验证
- 编码解码算法测试
- 控制场直径计算验证
- PID控制器参数测试
- 波束偏差计算验证

### 3. 红外制导系统测试

- 红外图像处理算法验证
- 目标识别和分类测试
- 运动轨迹预测验证
- 制导指令生成测试
- 抗干扰能力验证

### 4. 毫米波制导测试

- 雷达信号处理算法验证
- SwerlingRCS模型测试
- 目标检测算法验证

- 距离和速度测量测试
- 多目标跟踪验证

#### 5. 复合制导测试

- 制导模式切换算法验证
- 数据融合算法测试
- 制导系统激活触发器验证
- 制导质量评估测试
- 失效处理机制验证

### 5.1.4 状态信息系统测试

#### 1. ElementStatusInfo测试

- 统一状态信息结构验证
- GetStatusInfo方法实现测试
- 扩展属性字典功能验证
- 嵌套状态信息测试
- 时间戳同步验证

#### 2. 状态信息层次测试

- SimulationElement基础状态测试
- BaseMissile导弹状态验证
- BaseGuidanceSystem制导状态测试
- BaseEquipment设备状态验证
- BaseSensor传感器状态测试

#### 3. 状态信息格式化测试

- 状态信息序列化验证
- JSON格式输出测试
- 状态信息可读性验证
- 调试信息完整性测试
- 性能监控数据验证

### 5.1.5 事件系统测试

#### 1. 事件发布订阅测试

- 类型安全事件处理验证
- 事件订阅和取消订阅测试
- 事件发布和分发验证
- 异常隔离机制测试
- 事件队列管理验证

#### 2. 导弹生命周期事件测试

- 导弹发射事件验证
- 飞行阶段变化事件测试
- 制导状态变化事件验证
- 导弹爆炸事件测试
- 自毁事件验证

#### 3. 制导相关事件测试

- 激光照射事件验证
- 激光波束事件测试
- 红外制导指令事件验证
- 毫米波探测事件测试
- 目标命中事件验证

## 5.2 集成测试验证

### 5.2.1 系统集成测试

#### 1. 数据管理集成测试

- 配置文件加载与实体创建集成
- 数据缓存与仿真运行集成

- 错误处理与系统稳定性集成
- 热更新与运行时配置集成
- 多设备类型协同工作测试

## 2. 仿真管理集成测试

- 仿真状态控制与实体管理集成
- 时间同步与实体更新集成
- 天气系统与制导系统集成
- 事件系统与仿真流程集成
- 外部适配器与内部系统集成

## 3. 制导系统集成测试

- 制导系统与导弹基类集成
- 传感器与制导算法集成
- 制导事件与系统响应集成
- 干扰系统与制导性能集成
- 多制导系统协同工作测试

## 5.2.2 端到端测试

### 1. 完整攻击流程测试

- 导弹发射到命中全流程验证
- 制导系统激活和切换测试
- 目标探测和跟踪验证
- 制导控制和轨迹修正测试
- 命中判定和毁伤评估验证

### 2. 复杂场景测试

- 多导弹协同攻击测试
- 复杂环境条件下的制导验证
- 干扰条件下的系统性能测试
- 目标机动情况下的跟踪验证
- 系统故障情况下的容错测试

### 3. 性能压力测试

- 大量实体同时仿真测试
- 高频率事件处理验证
- 内存使用和泄漏检测
- CPU性能和优化验证
- 并发访问和线程安全测试

## 5.3 算法验证

### 5.3.1 制导算法验证

#### 1. 比例导引算法验证

- 导航比例系数优化测试
- 视线角速度计算精度验证
- 制导加速度收敛性测试
- 目标拦截轨迹分析
- 制导精度统计验证

#### 2. 目标跟踪算法验证

- 卡尔曼滤波跟踪精度测试
- 目标状态估计误差分析
- 运动轨迹预测准确性验证
- 多目标跟踪性能测试
- 跟踪丢失恢复能力验证

#### 3. 传感器算法验证

- 激光功率分布计算验证
- 红外图像处理算法测试
- 毫米波信号处理验证
- 多传感器融合算法测试

- 传感器噪声处理验证

### 5.3.2 物理模型验证

#### 1. 运动学模型验证

- 位置、速度、加速度关系验证
- 坐标系转换正确性测试
- 欧拉角与旋转矩阵一致性验证
- 运动状态积分精度测试
- 数值稳定性分析

#### 2. 升力模型验证

- 升力系数实验数据对比
- 俯仰角与升力关系验证
- 升力平衡点准确性测试
- 升力加速度计算验证
- 飞行阶段升力变化测试

#### 3. 传播模型验证

- 激光传播衰减模型验证
- 红外辐射传播模型测试
- 毫米波传播特性验证
- 大气影响模型准确性测试
- 环境条件影响分析

### 5.3.3 概率模型验证

#### 1. 命中概率模型验证

- 基础概率设定合理性验证
- 距离影响因子准确性测试
- 制导精度影响分析
- 环境条件影响验证
- 概率计算算法测试

#### 2. SwerlingRCS模型验证

- 各型Swerling模型实现验证
- RCS统计特性准确性测试
- 检测概率计算验证
- 目标起伏特性建模测试
- 雷达检测性能分析

#### 3. 随机噪声模型验证

- 运动状态噪声系数验证
- 飞行阶段噪声变化测试
- 噪声分布特性验证
- 噪声对制导精度影响分析
- 噪声模型参数优化

## 5.4 性能验证

### 5.4.1 计算性能验证

#### 1. 实时性能测试

- 仿真步长执行时间测量
- 制导算法计算时间分析
- 事件处理响应时间测试
- 状态更新性能验证
- 实时性要求满足度评估

#### 2. 内存性能测试

- 内存使用量监控
- 内存泄漏检测
- 缓存效率分析

- 垃圾回收影响测试
- 内存优化效果验证

### 3. 并发性能测试

- 多线程访问安全性验证
- 锁竞争和死锁检测
- 并发处理能力测试
- 线程池性能分析
- 并发优化效果验证

## 5.4.2 精度性能验证

### 1. 制导精度验证

- 各制导系统精度统计
- 制导误差分布分析
- 精度影响因素识别
- 精度改进效果验证
- 精度要求满足度评估

### 2. 数值精度验证

- 浮点数计算精度测试
- 数值积分误差分析
- 累积误差控制验证
- 数值稳定性测试
- 精度损失原因分析

### 3. 时间精度验证

- 时间同步精度测试
- 事件时间戳准确性验证
- 仿真时间与实际时间对比
- 时间步长影响分析
- 时间精度要求满足度评估

## 5.4.3 可靠性验证

### 1. 系统稳定性测试

- 长时间运行稳定性验证
- 异常情况处理能力测试
- 系统恢复能力验证
- 错误传播控制测试
- 系统容错能力分析

### 2. 数据一致性验证

- 配置数据一致性检查
- 状态数据同步验证
- 事件数据完整性测试
- 数据备份和恢复验证
- 数据损坏检测和修复

### 3. 接口兼容性验证

- 版本兼容性测试
- 接口向后兼容验证
- 外部系统集成测试
- 配置文件格式兼容性验证
- API接口稳定性测试

## 5.5 验证工具和方法

### 5.5.1 自动化测试工具

#### 1. 单元测试框架

- NUnit测试框架应用
- 测试用例自动生成

- 测试覆盖率分析
- 持续集成测试
- 回归测试自动化

## 2. 性能测试工具

- 性能监控工具集成
- 基准测试自动化
- 性能回归检测
- 性能瓶颈分析
- 性能优化验证

## 3. 集成测试工具

- 端到端测试自动化
- 场景测试脚本
- 数据驱动测试
- 环境配置自动化
- 测试结果分析

## 5.5.2 验证数据管理

### 1. 测试数据准备

- 标准测试数据集
- 边界条件测试数据
- 异常情况测试数据
- 性能测试数据
- 回归测试数据

### 2. 验证结果管理

- 测试结果记录和分析
- 验证报告自动生成
- 问题跟踪和管理
- 验证历史记录
- 质量指标统计

### 3. 基准数据维护

- 算法基准结果维护
- 性能基准数据更新
- 精度基准标准制定
- 基准数据版本管理
- 基准比较分析

## 5.5.3 验证流程管理

### 1. 验证计划制定

- 验证需求分析
- 验证策略制定
- 验证计划编制
- 资源分配和调度
- 风险评估和控制

### 2. 验证执行管理

- 验证任务分配
- 验证进度跟踪
- 问题识别和处理
- 验证质量控制
- 验证结果评审

### 3. 验证改进管理

- 验证效果评估
- 验证方法改进
- 验证工具优化
- 验证流程完善
- 经验总结 and 分享

## 6. 仿真接口

### 6.1 核心接口定义

#### 6.1.1 仿真管理接口

##### 1. ISimulationManager接口

仿真管理器的核心接口，提供仿真控制功能：

- StartSimulation(double timeStep): 启动仿真，指定时间步长
- PauseSimulation(): 暂停仿真
- ResumeSimulation(): 恢复仿真
- StopSimulation(): 停止仿真并重置
- UpdateSimulation(): 执行一次仿真更新
- 状态属性: CurrentTime、SimulationState、TimeStep

##### 2. ISimulationElement接口

仿真实体的基础接口，所有仿真对象必须实现：

- Update(double deltaTime): 更新实体状态
- GetStatusInfo(): 获取统一状态信息
- 属性访问: Id、KState、IsActive、ElementType
- 生命周期管理: Initialize()、Destroy()

##### 3. IEventSystem接口

事件系统接口，提供类型安全的事件处理：

- Subscribe(Action handler): 订阅特定类型事件
- Unsubscribe(Action handler): 取消订阅
- Publish(T eventData): 发布事件
- 事件队列管理和异常隔离

#### 6.1.2 数据管理接口

##### 1. IThreatSourceDataManager接口

数据管理器接口，提供配置数据访问：

- LoadData(): 加载所有配置数据
- GetMissileData(string modelName): 获取导弹配置
- GetIndicatorData(string modelName): 获取指示器配置
- GetSensorData(string modelName): 获取传感器配置
- GetEquipmentData(string modelName): 获取设备配置
- GetWeatherData(string modelName): 获取天气配置
- GetJammerData(string modelName): 获取干扰机配置

##### 2. IConfigurationData接口

配置数据的基础接口：

- Type: 设备类型标识
- Name: 多语言名称支持
- Properties: 基础属性字典
- Validate(): 配置数据验证

##### 3. IDataCache接口

数据缓存管理接口：

- Get(string key): 获取缓存数据
- Set(string key, T value): 设置缓存数据
- Remove(string key): 移除缓存数据
- Clear(): 清空缓存
- Contains(string key): 检查缓存存在性

#### 6.1.3 制导系统接口

##### 1. IGuidanceSystem接口

制导系统的统一接口：

- CalculateGuidanceAcceleration(): 计算制导加速度
- GetStatusInfo(): 获取制导系统状态
- IsGuidanceActive: 制导激活状态
- GuidanceAcceleration: 当前制导加速度
- 事件处理: ProcessGuidanceEvent(IGuidanceEvent)

## 2. IGuidanceEvent接口

制导事件的基础接口：

- SenderId: 事件发送者ID
- Timestamp: 事件时间戳
- EventType: 事件类型标识
- EventData: 事件数据载荷

## 3. ITargetTracker接口

目标跟踪器接口：

- TrackTarget(ITarget target): 跟踪目标
- PredictTargetPosition(double time): 预测目标位置
- GetTrackingAccuracy(): 获取跟踪精度
- IsTargetLocked: 目标锁定状态

# 6.1.4 传感器系统接口

## 1. ISensor接口

传感器的基础接口：

- DetectTargets(): 探测目标
- GetSensorData(): 获取传感器数据
- GetStatusInfo(): 获取传感器状态
- IsJammed: 干扰状态
- DetectionRange: 探测距离

## 2. ILaserSensor接口

激光传感器专用接口：

- CalculateLaserPower(Vector3 position): 计算激光功率
- GetQuadrantPowers(): 获取四象限功率
- LaserWavelength: 激光波长
- DetectorDiameter: 探测器直径

## 3. IRadarSensor接口

雷达传感器专用接口：

- CalculateRCS(ITarget target): 计算目标RCS
- GetSwerlingModel(): 获取Swerling模型
- WorkingFrequency: 工作频率
- TransmitPower: 发射功率

# 6.2 外部系统集成接口

## 6.2.1 仿真适配器接口

### 1. ISimulationAdapter接口

外部仿真系统适配器的统一接口：

- Initialize(): 初始化适配器
- OnTimeSync(double currentTime): 时间同步通知
- OnEntityUpdate(ISimulationElement entity): 实体状态更新通知
- OnEventPublished(T eventData): 事件发布通知
- SendExternalEvent(T eventData): 发送外部事件
- IsConnected: 连接状态

### 2. IUnityAdapter接口

Unity引擎适配器专用接口：



- UpdateUnityTransform(string entityId, Transform transform): 更新Unity变换
- CreateUnityGameObject(ISimulationElement entity): 创建Unity对象
- DestroyUnityGameObject(string entityId): 销毁Unity对象
- SyncUnityPhysics(): 同步Unity物理系统

### 3. IUnrealAdapter接口

Unreal引擎适配器专用接口:

- UpdateUnrealActor(string entityId, FTransform transform): 更新Unreal Actor
- SpawnUnrealActor(ISimulationElement entity): 生成Unreal Actor
- DestroyUnrealActor(string entityId): 销毁Unreal Actor
- SyncUnrealWorld(): 同步Unreal世界状态

## 6.2.2 数据交换接口

### 1. IDataExporter接口

数据导出接口:

- ExportSimulationData(string filePath): 导出仿真数据
- ExportEntityStates(IEnumerable entities): 导出实体状态
- ExportEventHistory(IEnumerable events): 导出事件历史
- SupportedFormats: 支持的导出格式

### 2. IDataImporter接口

数据导入接口:

- ImportSimulationData(string filePath): 导入仿真数据
- ImportEntityStates(string filePath): 导入实体状态
- ImportEventHistory(string filePath): 导入事件历史
- ValidateImportData(string filePath): 验证导入数据

### 3. IRealtimeDataStream接口

实时数据流接口:

- StartDataStream(): 启动数据流
- StopDataStream(): 停止数据流
- SendRealtimeData(T data): 发送实时数据
- OnDataReceived(Action handler): 数据接收处理
- StreamStatus: 数据流状态

## 6.2.3 网络通信接口

### 1. INetworkManager接口

网络管理器接口:

- StartServer(int port): 启动服务器
- ConnectToServer(string address, int port): 连接服务器
- SendMessage(T message): 发送消息
- OnMessageReceived(Action handler): 消息接收处理
- IsConnected: 连接状态

### 2. IMessageSerializer接口

消息序列化接口:

- Serialize(T obj): 序列化对象
- Deserialize(byte[] data): 反序列化对象
- GetSupportedTypes(): 获取支持的类型
- SerializationFormat: 序列化格式

## 6.3 状态信息接口

### 6.3.1 统一状态信息

#### 1. ElementStatusInfo类

统一的状态信息结构:

- Id: 实体唯一标识符
- ElementType: 实体类型名称
- KState: 运动学状态信息
- IsActive: 活动状态标志
- Timestamp: 状态时间戳
- ExtendedProperties: 扩展属性字典

## 2. 状态信息扩展属性

不同实体类型的扩展属性:

- 导弹状态: 飞行时间、飞行阶段、制导状态、升力加速度
- 制导系统状态: 制导加速度、干扰状态、目标锁定状态
- 传感器状态: 探测状态、干扰功率、传感器数据
- 设备状态: 生命值、属性信息、工作状态

## 3. 状态信息格式化

状态信息的标准化输出:

- JSON格式序列化
- 可读性格式化
- 调试信息输出
- 性能监控数据

## 6.3.2 状态查询接口

### 1. IStatusProvider接口

状态提供者接口:

- GetStatusInfo(): 获取当前状态信息
- GetHistoryStatus(DateTime timestamp): 获取历史状态
- GetStatusSummary(): 获取状态摘要
- SubscribeStatusUpdates(Action handler): 订阅状态更新

### 2. IStatusCollector接口

状态收集器接口:

- CollectAllStatus(): 收集所有实体状态
- CollectStatusByType(string elementType): 按类型收集状态
- CollectStatusByFilter(Func<ISimulationElement, bool> filter): 按条件收集状态
- GetStatusStatistics(): 获取状态统计信息

## 6.4 配置接口

### 6.4.1 配置管理接口

#### 1. IConfigurationManager接口

配置管理器接口:

- LoadConfiguration(string configPath): 加载配置
- SaveConfiguration(string configPath): 保存配置
- GetConfigurationValue(string key): 获取配置值
- SetConfigurationValue(string key, T value): 设置配置值
- ValidateConfiguration(): 验证配置

#### 2. IConfigurationProvider接口

配置提供者接口:

- GetConfiguration(string key): 获取特定配置
- GetAllConfigurations(): 获取所有配置
- IsConfigurationAvailable(string key): 检查配置可用性
- GetConfigurationMetadata(string key): 获取配置元数据

### 6.4.2 动态配置接口

#### 1. IHotReloadable接口

热重载接口:

- ReloadConfiguration(): 重新加载配置
- OnConfigurationChanged(Action handler): 配置变更通知
- CanHotReload: 是否支持热重载
- LastReloadTime: 最后重载时间

## 2. IConfigurationValidator接口

配置验证器接口:

- ValidateConfiguration(object config): 验证配置
- GetValidationRules(): 获取验证规则
- GetValidationErrors(): 获取验证错误
- IsValid: 配置是否有效

## 6.5 扩展接口

### 6.5.1 插件系统接口

#### 1. IPlugin接口

插件基础接口:

- Initialize(IServiceProvider serviceProvider): 初始化插件
- Start(): 启动插件
- Stop(): 停止插件
- GetPluginInfo(): 获取插件信息
- IsEnabled: 插件启用状态

#### 2. IPluginManager接口

插件管理器接口:

- LoadPlugin(string pluginPath): 加载插件
- UnloadPlugin(string pluginName): 卸载插件
- GetLoadedPlugins(): 获取已加载插件
- EnablePlugin(string pluginName): 启用插件
- DisablePlugin(string pluginName): 禁用插件

### 6.5.2 自定义扩展接口

#### 1. ICustomGuidanceSystem接口

自定义制导系统接口:

- RegisterCustomGuidance(string name, Type guidanceType): 注册自定义制导
- CreateCustomGuidance(string name): 创建自定义制导实例
- GetSupportedGuidanceTypes(): 获取支持的制导类型

#### 2. ICustomSensor接口

自定义传感器接口:

- RegisterCustomSensor(string name, Type sensorType): 注册自定义传感器
- CreateCustomSensor(string name): 创建自定义传感器实例
- GetSupportedSensorTypes(): 获取支持的传感器类型

## 6.6 接口使用示例

### 6.6.1 基本仿真控制

#### 1. 仿真初始化和启动

基本的仿真控制流程:

- 创建仿真管理器实例
- 加载配置数据
- 注册仿真实体
- 启动仿真循环
- 处理仿真事件

#### 2. 实体创建和管理

仿真实体的创建和管理：

- 从配置数据创建实体
- 注册实体到仿真管理器
- 设置实体初始状态
- 订阅实体事件
- 管理实体生命周期

## 6.6.2 外部系统集成

### 1. Unity集成示例

与Unity引擎的集成：

- 创建Unity适配器
- 同步仿真时间
- 更新Unity对象变换
- 处理Unity事件
- 同步物理系统

### 2. 数据导出示例

仿真数据的导出：

- 配置数据导出器
- 选择导出格式
- 导出实体状态
- 导出事件历史
- 验证导出数据

## 6.6.3 自定义扩展

### 1. 自定义制导系统

实现自定义制导算法：

- 继承IGuidanceSystem接口
- 实现制导算法逻辑
- 注册自定义制导类型
- 配置制导参数
- 测试制导性能

### 2. 自定义传感器

实现自定义传感器：

- 继承ISensor接口
- 实现传感器算法
- 注册自定义传感器类型
- 配置传感器参数
- 验证传感器功能

# 7. 附录

## 7.1 术语表

### 7.1.1 基础术语

**仿真实体 (Simulation Element)**

- 仿真系统中的基本对象，包括导弹、目标、传感器等
- 继承自SimulationElement基类
- 具有统一的生命周期管理和状态信息

**运动学状态 (Kinematic State)**

- 描述实体运动状态的数据结构
- 包含位置、朝向、速度等信息
- 使用KinematicState类统一管理

## 制导系统 (Guidance System)

- 控制导弹飞行轨迹的系统
- 包括激光、红外、毫米波等多种制导方式
- 实现IGuidanceSystem接口

## 事件系统 (Event System)

- 基于发布订阅模式的消息传递机制
- 提供类型安全的事件处理
- 支持异步事件处理和异常隔离

## 7.1.2 制导术语

### 比例导引 (Proportional Navigation)

- 经典的制导算法
- 基于视线角速度的制导方法
- 广泛应用于各种制导系统

### 四象限探测器 (Quadrant Detector)

- 激光制导中的关键传感器
- 将探测区域分为四个象限
- 通过功率分布确定目标方位

### SwerlingRCS模型

- 雷达散射截面的统计模型
- 包括Swerling I、II、III、IV型
- 用于毫米波制导的目标建模

### 升力模型 (Lift Model)

- 基于俯仰角计算升力加速度
- 升力平衡点为5°俯仰角
- 升力系数为1.0 m/s<sup>2</sup>/度

## 7.1.3 技术术语

### 智能路径解析 (Smart Path Resolution)

- ThreatSourceDataManager的核心功能
- 自动查找DLL上级目录的data文件夹
- 提供回退路径机制

### 复合制导 (Composite Guidance)

- 集成多种制导方式的系统
- 支持串行和并行制导模式
- 提供制导冗余和性能优化

### 状态信息管理 (Status Information Management)

- ElementStatusInfo统一状态信息结构
- GetStatusInfo方法的标准实现
- 支持扩展属性和嵌套状态

### TOML配置格式

- Tom's Obvious, Minimal Language
- 人类可读的配置文件格式
- 支持类型安全和嵌套结构

## 7.1.4 坐标系术语

### 世界坐标系 (World Coordinate System)

- 右手坐标系
- X轴指向东方, Y轴指向上方, Z轴指向北方
- 仿真场景的全局坐标系

### 本体坐标系 (Body Coordinate System)

- 导弹本体的局部坐标系
- X轴指向右翼, Y轴指向上方, Z轴指向后方
- 前向方向为-Z轴 (重要变更)

### 欧拉角 (Euler Angles)

- 偏航角 (Yaw) : 绕Y轴旋转
- 俯仰角 (Pitch) : 绕X轴旋转
- 滚转角 (Roll) : 绕Z轴旋转

## 7.2 技术参数

### 7.2.1 系统性能参数

#### 仿真性能

- 最大实体数量: 1000+
- 仿真步长: 0.001-1.0秒
- 事件处理延迟: <1ms
- 内存使用: <500MB (1000实体)

#### 制导精度

- 激光半主动制导: CEP < 1m
- 激光驾束制导: CEP < 0.5m
- 红外成像制导: CEP < 2m
- 毫米波制导: CEP < 3m

#### 数据处理性能

- 配置文件加载: <100ms
- 状态信息更新: <10 $\mu$ s/实体
- 事件处理: <1 $\mu$ s/事件
- 数据缓存命中率: >95%

### 7.2.2 物理模型参数

#### 升力模型参数

- 有效俯仰角范围: -5°到15°
- 升力平衡点: 5°俯仰角
- 升力系数: 1.0 m/s<sup>2</sup>/度
- 重力加速度: 9.81 m/s<sup>2</sup>

#### 传播模型参数

- 激光大气透过率: 0.7-0.9
- 红外大气透过率: 0.6-0.8
- 毫米波降雨衰减: 0.1-10 dB/km
- 视距传播距离: 50km+

#### 噪声模型参数

- Launch阶段噪声系数: 0.1
- Cruise阶段噪声系数: 0.05
- TerminalGuidance阶段噪声系数: 0.02
- 噪声分布: 高斯分布

### 7.2.3 配置参数范围

#### 导弹参数范围

- 最大速度: 100-2000 m/s
- 最大加速度: 10-100 m/s<sup>2</sup>
- 最大飞行时间: 10-300秒
- 最大飞行距离: 1-50公里

#### 制导参数范围

- 导航比例系数：2-6
- 最大制导加速度：20-80 m/s<sup>2</sup>
- 制导激活距离：100-5000米
- 制导精度要求：0.1-5米

#### 传感器参数范围

- 激光功率：1-100瓦
- 探测距离：100-10000米
- 视场角：1-30度
- 工作频率：1-100 GHz

## 7.3 配置示例

### 7.3.1 导弹配置示例

```
Type = "missile"

[Name]
zh = "激光半主动制导导弹"
en = "Laser Semi-Active Guided Missile"

[Properties]
MaxSpeed = 800.0
MaxAcceleration = 50.0
MaxFlightTime = 120.0
MaxFlightDistance = 8000.0
CruiseAttackAngle = 5.0
GuidanceDownwardViewingAngle = 15.0

[LaserSemiActiveGuidanceConfig]
DetectorDiameter = 0.1
FieldOfView = 30.0
LockThreshold = 0.001
MaxGuidanceAcceleration = 40.0
NavigationGain = 3.0
```

### 7.3.2 复合制导配置示例

```

Type = "missile"

[Name]
zh = "毫米波红外复合制导导弹"
en = "MMW/IR Composite Guided Missile"

[[GuidanceSuite]]
Name = "毫米波制导"
GuidanceSystemType = "millimeterwave"
ActivationTrigger = "LaunchTime"
MaxAcquisitionGuidanceTime = 10.0
MinStableGuidanceTime = 2.0
ContinueOnFailure = true

[[GuidanceSuite]]
Name = "红外成像制导"
GuidanceSystemType = "infraredimagingterminalguidance"
ActivationTrigger = "PreviousComplete"
MaxAcquisitionGuidanceTime = 15.0
MinStableGuidanceTime = 3.0
ContinueOnFailure = false

[MillimeterWaveGuidanceConfig]
WorkingFrequency = 35.0
TransmitPower = 10.0
MaxDetectionRange = 5000.0

[InfraredImagingGuidanceConfig]
MaxDetectionRange = 3000.0
FieldOfView = 20.0
TargetRecognitionProbability = 0.9

```

### 7.3.3 传感器配置示例

```

Type = "sensor"

[Name]
zh = "毫米波雷达"
en = "Millimeter Wave Radar"

[Properties]
WorkingFrequency = 35.0
TransmitPower = 15.0
AntennaGain = 30.0
MaxDetectionRange = 8000.0
FieldOfView = 45.0
SwerlingModel = "SwerlingII"

```

## 7.4 开发指南

### 7.4.1 开发环境要求

#### 软件环境

- .NET 8.0+
- Visual Studio 2019+
- NUnit 3.12+
- Tomlyn 0.16+

#### 硬件环境

- CPU: Intel i5 8代+
- 内存: 8GB+
- 存储: SSD 100GB+



- 显卡：支持DirectX 11+

#### 开发工具

- Git版本控制
- NuGet包管理
- MSBuild构建系统
- NUnit测试框架

### 7.4.2 编码规范

#### 命名规范

- 类名：PascalCase（如BaseMissile）
- 方法名：PascalCase（如GetStatusInfo）
- 属性名：PascalCase（如IsActive）
- 字段名：camelCase（如isGuidanceActive）
- 常量名：UPPER\_CASE（如MAX\_SPEED）

#### 注释规范

- 类注释：描述类的用途和功能
- 方法注释：描述方法的功能、参数和返回值
- 复杂逻辑注释：解释算法和业务逻辑
- 注释语言：中文

#### 代码结构

- 单一职责原则
- 开闭原则
- 依赖倒置原则
- 接口隔离原则

### 7.4.3 测试指南

#### 单元测试

- 测试覆盖率：>80%
- 测试命名：Should\_ExpectedBehavior\_When\_StateUnderTest
- 测试结构：Arrange-Act-Assert
- 测试数据：使用测试数据构建器

#### 集成测试

- 端到端测试场景
- 外部系统集成测试
- 性能基准测试
- 压力测试

#### 测试自动化

- 持续集成测试
- 回归测试自动化
- 性能回归检测
- 测试报告生成

## 7.5 故障排除

### 7.5.1 常见问题

#### 配置文件问题

- 问题：配置文件加载失败
- 原因：文件路径错误或格式错误
- 解决：检查路径解析日志，验证TOML格式

#### 制导系统问题

- 问题：制导不生效
- 原因：制导事件未接收或参数错误

- 解决：检查事件订阅，验证制导参数

#### 性能问题

- 问题：仿真运行缓慢
- 原因：实体数量过多或算法复杂度高
- 解决：优化算法，减少实体数量

## 7.5.2 调试技巧

#### 日志分析

- 启用详细日志输出
- 分析错误堆栈信息
- 跟踪关键方法调用
- 监控性能指标

#### 状态检查

- 使用GetStatusInfo方法
- 检查实体状态信息
- 验证制导系统状态
- 监控事件流

#### 性能分析

- 使用性能分析工具
- 监控内存使用
- 分析CPU占用
- 检查线程安全

# 第三方引擎集成示例

本目录包含了将ThreatSource仿真系统与第三方引擎集成的示例代码。这些示例展示了如何使用适配器模式将不同的游戏引擎与仿真系统进行集成。

## 系统要求

- .NET 8.0 或更高版本
- ThreatSource.dll 库文件
- 对应的第三方引擎开发环境

## 示例文件

### UEExample.cs

虚幻引擎(Unreal Engine)集成示例，展示了：

- 虚幻引擎与ThreatSource仿真系统的双向通信
- Actor与仿真实体的映射和同步
- 事件的发布和订阅
- 坐标系转换和数据适配

## 核心功能

```

using ThreatSource.Simulation;
using ThreatSource.Data;
using ThreatSource.Missile;

public class UnrealThreatSourceAdapter
{
    private ISimulationManager _simulationManager;
    private ThreatSourceDataManager _dataManager;
    private Dictionary<string, object> _entityActors;

    public void UpdateSimulation()
    {
        // 更新仿真
        _simulationManager.UpdateSimulation();

        // 同步实体状态到虚幻引擎Actor
        SyncEntityStates();
    }

    public string CreateMissile(string missileType, Vector3 location, string targetId)
    {
        // 从配置创建导弹
        var missileData = _dataManager.GetMissile(missileType);
        var missile = new InfraredImagingTerminalGuidanceMissile(missileId, missileData);

        // 在虚幻引擎中创建Actor
        var missileActor = _unrealEngine.SpawnActor("BP_Missile", location, rotation);

        // 建立映射关系
        _entityActors[missileId] = missileActor;

        return missileId;
    }
}

```

## UnityExample.cs - 方法一示例

适用于：直接引用.NET库方式

Unity引擎集成示例，展示了：

- Unity引擎与ThreatSource仿真系统的双向通信
- GameObject与仿真实体的映射和转换
- MonoBehaviour生命周期管理
- 事件系统的使用和视觉效果处理

使用前提：

- 已将 ThreatSource.dll 复制到 Assets/Plugins/
- 配置文件已放置在 Assets/StreamingAssets/ThreatSource/

## 核心功能

```

using ThreatSource.Simulation;
using ThreatSource.Data;
using ThreatSource.Missile;
using UnityEngine;

public class UnityThreatSourceAdapter : MonoBehaviour
{
    private ISimulationManager _simulationManager;
    private ThreatSourceDataManager _dataManager;
    private Dictionary<string, GameObject> _entityGameObjects;

    private void Update()
    {
        // 更新仿真
        _simulationManager.UpdateSimulation();

        // 同步实体状态到Unity GameObject
        SyncEntityStates();
    }

    public void CreateMissile(string missileType, Vector3 position, string targetId)
    {
        // 从配置创建导弹
        var missileData = _dataManager.GetMissile(missileType);
        var missile = new InfraredImagingTerminalGuidanceMissile(missileId, missileData);

        // 在Unity中创建GameObject
        var missileGameObject = Instantiate(missilePrefab, position, Quaternion.identity);

        // 建立映射关系
        _entityGameObjects[missileId] = missileGameObject;
    }
}

```

## UnityPackageExample.cs - 方法二示例

适用于：**Unity Package**方式

Unity Package集成示例，展示了：

- Package级别的管理和配置
- ThreatSourceUnity命名空间的使用
- Package专用的事件系统和API
- Assembly Definition环境下的集成

使用前提：

- 已按照方法二创建完整的Package结构
- 配置了正确的package.json和Assembly Definition
- 通过Package Manager安装或本地引用Package

## Package核心功能

```
using ThreatSourceUnity;

public class PackageExampleUsage : MonoBehaviour
{
    private void Start()
    {
        var packageManager = ThreatSourcePackageManager.Instance;

        // 创建目标
        string targetId = packageManager.CreateTarget(
            "package_target_001",
            new Vector3(1000, 0, 100),
            new Vector3(-50, 0, 0)
        );

        // 创建导弹
        string missileId = packageManager.CreateMissile(
            "IR_Missile_Example",
            Vector3.zero,
            targetId
        );
    }
}
```

## Unity集成方式

Unity引擎与ThreatSource仿真库提供两种主要的集成方式，开发者可以根据项目需求和团队情况选择合适的方案。

### 方法一：直接引用.NET库（推荐）

适用场景：

- 快速原型开发
- 单一项目使用
- 需要完整仿真功能访问

操作步骤：

#### 1. 下载并解压库文件

```
# 下载 ThreatSourceLibrary-{version}.zip
# 解压到临时目录
```

#### 2. 复制核心库文件

```
# 将以下文件复制到 Unity 项目
ThreatSource.dll → Assets/Plugins/
ThreatSource.deps.json → Assets/Plugins/
ThreatSource.xml → Assets/Plugins/（可选，用于智能提示）
```

#### 3. 复制配置文件

```
# 将配置目录复制到 StreamingAssets
data/ → Assets/StreamingAssets/ThreatSource/data/
```

#### 4. 配置Unity项目设置

```
// 在 Project Settings 中设置：
// - Api Compatibility Level: .NET Standard 2.1 或更高
// - Scripting Backend: Mono 或 IL2CPP
```

#### 5. 验证集成

```

using ThreatSource.Simulation;
using ThreatSource.Data;

public class ThreatSourceTest : MonoBehaviour
{
    private void Start()
    {
        // 测试库是否正确加载
        var dataManager = new ThreatSourceDataManager();
        Debug.Log("ThreatSource库加载成功! ");
    }
}

```

#### 优势:

- 实现简单，快速上手
- 直接访问完整API
- 性能最优，无额外包装
- 易于调试和问题排查

#### 劣势:

- 文件管理相对分散
- 版本更新需手动替换文件
- 难以在多项目间共享

## 方法二：创建Unity Package

#### 适用场景:

- 团队协作开发
- 多项目复用
- 需要版本管理
- 计划发布到Package Registry

#### 创建步骤:

##### 1. 创建Package目录结构

```

Assets/ThreatSourceUnity/
├── package.json
├── README.md
├── CHANGELOG.md
├── Runtime/
│   ├── ThreatSourceUnity.asmdef
│   ├── Scripts/
│   │   ├── UnityThreatSourceAdapter.cs
│   │   ├── ThreatSourceManager.cs
│   │   └── Utils/
│   │       ├── CoordinateConverter.cs
│   │       └── ConfigLoader.cs
│   └── Plugins/
│       ├── ThreatSource.dll
│       └── ThreatSource.deps.json
├── Editor/
│   ├── ThreatSourceUnity.Editor.asmdef
│   └── ThreatSourceSetup.cs
└── Samples~/
    └── BasicExample/
        └── ThreatSourceExample.cs

```

##### 2. 配置package.json

```
{
  "name": "com.yourcompany.threatsource",
  "version": "1.1.22",
  "displayName": "ThreatSource Library",
  "description": "军事威胁源仿真库Unity集成包",
  "unity": "2020.3",
  "dependencies": {
    "com.unity.mathematics": "1.2.6"
  },
  "keywords": ["simulation", "missile", "guidance"],
  "author": {
    "name": "Your Company",
    "email": "support@company.com"
  }
}
```

### 3. 创建Assembly Definition

```
// Runtime/ThreatSourceUnity.asmdef
{
  "name": "ThreatSourceUnity",
  "rootNamespace": "ThreatSourceUnity",
  "references": ["Unity.Mathematics"],
  "includePlatforms": [],
  "excludePlatforms": [],
  "allowUnsafeCode": false
}
```

### 4. 创建编辑器工具



```

// Editor/ThreatSourceSetup.cs
using UnityEngine;
using UnityEditor;
using System.IO;

public class ThreatSourceSetup : EditorWindow
{
    [MenuItem("Tools/ThreatSource/Package Setup")]
    static void ShowWindow()
    {
        GetWindow<ThreatSourceSetup>("ThreatSource Setup");
    }

    private void OnGUI()
    {
        GUILayout.Label("ThreatSource Package Configuration", EditorStyles.boldLabel);

        if (GUILayout.Button("Setup StreamingAssets"))
        {
            SetupStreamingAssets();
        }

        if (GUILayout.Button("Validate Configuration"))
        {
            ValidateSetup();
        }
    }

    private void SetupStreamingAssets()
    {
        string targetPath = "Assets/StreamingAssets/ThreatSource";
        if (!Directory.Exists(targetPath))
        {
            Directory.CreateDirectory(targetPath);
            Debug.Log($"Created directory: {targetPath}");
        }
    }

    private void ValidateSetup()
    {
        // 验证配置完整性
        bool isValid = File.Exists("Assets/StreamingAssets/ThreatSource/data");
        Debug.Log($"Setup validation: {(isValid ? "PASSED" : "FAILED")}");
    }
}

```

## 5. 添加示例代码

```
// Samples~/BasicExample/ThreatSourceExample.cs
using UnityEngine;
using ThreatSource.Simulation;
using ThreatSource.Data;

namespace ThreatSourceUnity.Samples
{
    public class ThreatSourceExample : MonoBehaviour
    {
        private UnityThreatSourceAdapter adapter;

        private void Start()
        {
            adapter = GetComponent<UnityThreatSourceAdapter>();
            if (adapter == null)
            {
                adapter = gameObject.AddComponent<UnityThreatSourceAdapter>();
            }

            CreateExampleScene();
        }

        private void CreateExampleScene()
        {
            // 创建示例仿真场景
            adapter.CreateTarget("target1", new Vector3(1000, 0, 0), Vector3.zero);
            adapter.CreateMissile("Hellfire", Vector3.zero, "target1");
        }
    }
}
```

#### 优势:

- 更好的项目组织和结构
- 便于版本管理和更新
- 支持团队协作和共享
- 可发布到Package Registry
- 包含完整的文档和示例

#### 劣势:

- 初始设置相对复杂
- 需要Package管理知识
- 可能增加项目复杂度

## 集成方式选择指导

因素      直接引用.NET库创建Unity Package

项目规模 小型、原型项目    中大型、生产项目

团队规模 个人、小团队      多人团队




版本管理 手动管理          Package Manager

复用需求 单项目使用      多项目复用

上手难度 简单              中等

维护成本低              中等

推荐选择:

-  快速原型/学习: 选择方法一
-  团队项目/生产环境: 选择方法二
-  从方法一迁移到方法二: 随项目成熟度逐步升级

## 示例文件对应关系

集成方式	示例文件	主要特点
方法一: 直接引用.NET库	<a href="#">UnityExample.cs</a>	直接使用ThreatSource命名空间, 简单直接
方法二: <b>Unity Package</b>	<a href="#">UnityPackageExample.cs</a>	使用ThreatSourceUnity命名空间, Package化管理

使用建议:

- 初学者或快速验证概念：参考 `UnityExample.cs`
- 正式项目开发：参考 `UnityPackageExample.cs`
- 两个示例文件都包含完整的集成代码，可直接复制使用

## 使用说明

### 基本集成流程

#### 1. 初始化仿真系统

```
_simulationManager = new SimulationManager();
_dataManager = new ThreatSourceDataManager();
_simulationManager.StartSimulation(timeStep);
```

#### 2. 订阅仿真事件

```
_simulationManager.Subscribe<MissileFireEvent>(OnMissileFireEvent);
_simulationManager.Subscribe<MissileExplodeEvent>(OnMissileExplodeEvent);
_simulationManager.Subscribe<FlightPhaseChangeEvent>(OnFlightPhaseChangeEvent);
```

#### 3. 创建实体映射

```
// 创建仿真实体
var missile = new InfraredImagingTerminalGuidanceMissile(id, data);
_simulationManager.RegisterEntity(missile);

// 创建引擎对象 (Unity GameObject 或 Unreal Actor)
var visualObject = CreateVisualObject(missile);

// 建立映射关系
_entityMappings[id] = visualObject;
```

#### 4. 同步状态

```
private void SyncEntityStates()
{
    foreach (var kvp in _entityMappings)
    {
        var entity = _simulationManager.GetEntity(kvp.Key);
        var visualObject = kvp.Value;

        // 同步位置和朝向
        SyncTransform(entity, visualObject);
    }
}
```

### 坐标系转换

#### Unity集成

```
// ThreatSource -> Unity 坐标转换
gameObject.transform.position = new Vector3(
    entity.KState.Position.X,
    entity.KState.Position.Z, // Unity使用Y作为高度
    entity.KState.Position.Y
);
```

#### 虚幻引擎集成

```
// ThreatSource -> Unreal 坐标转换 (右手系 -> 左手系)
var location = new Vector3(
    entity.KState.Position.X,
    -entity.KState.Position.Y, // Y轴翻转
    entity.KState.Position.Z
);
```

## 事件处理

### 导弹发射事件

```
private void OnMissileFireEvent(MissileFireEvent evt)
{
    // 播放发射特效
    var visualObject = GetVisualObject(evt.SenderId);
    PlayLaunchEffect(visualObject);
}
```

### 导弹爆炸事件

```
private void OnMissileExplodeEvent(MissileExplodeEvent evt)
{
    // 播放爆炸特效
    PlayExplosionEffect(evt.Position);

    // 销毁视觉对象
    DestroyVisualObject(evt.SenderId);
}
```

### 飞行阶段变化事件

```
private void OnFlightPhaseChangeEvent(FlightPhaseChangeEvent evt)
{
    var visualObject = GetVisualObject(evt.SenderId);

    switch (evt.NewPhase)
    {
        case FlightPhase.Boost:
            ShowBoostEffect(visualObject);
            break;
        case FlightPhase.Terminal:
            ShowTerminalEffect(visualObject);
            break;
    }
}
```

## 关键概念

### 适配器模式

适配器模式用于连接ThreatSource仿真系统和第三方引擎：

- 仿真层：ThreatSource核心仿真逻辑
- 适配器层：数据转换和事件映射
- 引擎层：Unity/Unreal等第三方引擎

### 实体映射

在仿真系统和游戏引擎之间建立实体对应关系：

```
// 映射关系管理
private Dictionary<string, VisualObject> _entityMappings;

// 创建映射
public void CreateEntityMapping(string entityId, VisualObject visualObject)
{
    _entityMappings[entityId] = visualObject;
}

// 同步状态
public void SyncEntity(string entityId)
{
    var entity = _simulationManager.GetEntity(entityId);
    var visualObject = _entityMappings[entityId];

    // 同步位置、朝向、状态等
    SyncTransform(entity, visualObject);
    SyncVisualState(entity, visualObject);
}
}
```

## 事件系统

统一的事件处理机制：

```
// 事件订阅
_simulationManager.Subscribe<TEvent>(handler);

// 事件处理
private void HandleEvent(TEvent evt)
{
    // 转换为引擎特定的操作
    ConvertToEngineOperation(evt);
}
}
```

## 性能优化

### 批量更新

```
// 批量同步实体状态，减少单次调用开销
private void BatchSyncEntities()
{
    var entities = _simulationManager.GetAllEntities();
    foreach (var entity in entities)
    {
        if (_entityMappings.TryGetValue(entity.Id, out var visualObject))
        {
            SyncEntity(entity, visualObject);
        }
    }
}
}
```

### 距离剔除

```
// 只同步视野范围内的实体
private void SyncVisibleEntities(Vector3 viewerPosition, float maxDistance)
{
    foreach (var kvp in _entityMappings)
    {
        var entity = _simulationManager.GetEntity(kvp.Key);
        var distance = Vector3.Distance(entity.KState.Position, viewerPosition);

        if (distance <= maxDistance)
        {
            SyncEntity(entity, kvp.Value);
        }
    }
}
```

## 帧率控制

```
// 控制仿真更新频率
private float _lastUpdateTime;
private float _updateInterval = 0.02f; // 50Hz

private void Update()
{
    if (Time.time - _lastUpdateTime >= _updateInterval)
    {
        _simulationManager.UpdateSimulation();
        SyncEntityStates();
        _lastUpdateTime = Time.time;
    }
}
```

## 注意事项

### 通用注意事项

1. **线程安全**: 确保仿真更新和引擎渲染在正确的线程中执行
2. **性能优化**: 合理控制同步频率, 避免过度更新
3. **资源管理**: 正确处理实体的创建和销毁
4. **异常处理**: 添加完整的异常处理机制

### Unity特定注意事项

1. **主线程操作**: Unity API必须在主线程中调用
2. **生命周期管理**: 正确处理MonoBehaviour的生命周期
3. **预制体管理**: 合理组织和加载预制体资源
4. **坐标系转换**: 注意Unity的左手坐标系

### 虚幻引擎特定注意事项

1. **蓝图集成**: 考虑与蓝图系统的集成
2. **Actor生命周期**: 正确管理Actor的创建和销毁
3. **坐标系转换**: 处理右手系到左手系的转换
4. **性能分析**: 使用虚幻引擎的性能分析工具

## 扩展功能

### 多人同步

```
// 网络同步支持
public class NetworkedThreatSourceAdapter
{
    public void SyncEntityOverNetwork(string entityId, EntityState state)
    {
        // 通过网络同步实体状态
        NetworkManager.SendEntityUpdate(entityId, state);
    }
}
```

## 录制回放

```
// 仿真录制和回放
public class SimulationRecorder
{
    public void RecordFrame(float timestamp, List<EntityState> states)
    {
        // 记录当前帧的所有实体状态
        _recordedFrames.Add(new SimulationFrame(timestamp, states));
    }

    public void PlaybackFrame(int frameIndex)
    {
        // 回放指定帧的状态
        var frame = _recordedFrames[frameIndex];
        ApplyFrameStates(frame);
    }
}
```

## 自定义渲染

```
// 自定义渲染效果
public class CustomMissileRenderer
{
    public void RenderMissileTrail(Vector3 position, Vector3 velocity)
    {
        // 根据导弹状态渲染自定义拖尾效果
        var trailLength = velocity.magnitude * 0.1f;
        RenderTrail(position, velocity.normalized, trailLength);
    }
}
```

# 导弹仿真示例

本目录包含了使用ThreatSource仿真系统进行导弹仿真的示例代码。这些示例展示了如何配置和运行不同类型的导弹仿真。

## 关于本库

ThreatSource 是一个基于 .NET 8.0 的类库，提供了完整的导弹仿真功能，包括：

- 多种制导系统（激光、红外、毫米波等）
- 运动学仿真和物理建模
- 事件驱动的仿真架构
- 智能数据管理系统
- 外部系统集成支持

## 系统要求

### C#/.NET 用户

- .NET 8.0 或更高版本
- 通过 NuGet 包管理器安装或直接引用 ThreatSource.dll
- Visual Studio 2019+ 或 Visual Studio Code

### C++用户

本库是一个 .NET 类库，C++用户需要通过 C++/CLI 包装层来使用：

- Windows 操作系统
- Visual Studio 2019 或更高版本（支持C++/CLI）
- .NET 8.0 运行时
- 创建 C++/CLI 项目并引用 ThreatSource.dll

## 示例文件

### C#示例 ([IRMissileSimulation.cs](#))

红外成像制导导弹仿真示例，展示了：

- 如何使用SimulationManager管理仿真
- 如何从配置文件创建导弹实体
- 如何设置目标和仿真环境
- 如何订阅和处理仿真事件
- 如何运行仿真循环并获取结果

### 核心代码片段



```

using ThreatSource.Simulation;
using ThreatSource.Data;
using ThreatSource.Missile;

public class IRMissileSimulationExample
{
    private ISimulationManager _simulationManager;
    private ThreatSourceDataManager _dataManager;

    public async Task RunSimulationExample()
    {
        // 1. 初始化仿真管理器
        _simulationManager = new SimulationManager();
        _dataManager = new ThreatSourceDataManager();
        _simulationManager.StartSimulation(0.02); // 20ms时间步长

        // 2. 创建目标实体
        var target = new BaseEquipment("target_001")
        {
            KState = new KinematicState
            {
                Position = new Vector3(5000, 0, 1000), // 5km距离
                Velocity = new Vector3(-50, 0, 0),      // 50m/s移动
                Orientation = new Orientation(0, 0, 0)
            }
        };

        // 3. 从配置创建导弹
        var missileData = _dataManager.GetMissile("IR_Missile_Example");
        var missile = new InfraredImagingTerminalGuidanceMissile("missile_001", missileData)
        {
            KState = new KinematicState
            {
                Position = new Vector3(0, 0, 100),
                Velocity = new Vector3(200, 0, 0),
                Orientation = new Orientation(0, 0, 0)
            }
        };

        // 4. 注册实体和事件
        _simulationManager.RegisterEntity(target);
        _simulationManager.RegisterEntity(missile);
        _simulationManager.Subscribe<MissileExplodeEvent>(OnMissileExplode);

        // 5. 运行仿真循环
        while (simulationTime < maxTime && missile.IsActive)
        {
            _simulationManager.UpdateSimulation();
            // 处理仿真状态...
        }
    }
}

```

## C++示例 (IRMissileSimulation.cpp)

这是一个使用C++/CLI的示例代码，展示了如何在C++项目中使用本库：

- 如何创建C++/CLI包装层
- 如何配置导弹实体和目标
- 如何设置仿真参数
- 如何运行仿真并获取结果

## 核心代码片段

```

using namespace ThreatSource::Simulation;
using namespace ThreatSource::Data;
using namespace ThreatSource::Missile;

public ref class IRMissileSimulationWrapper
{
private:
    ISimulationManager^ simulationManager;
    ThreatSourceDataManager^ dataManager;

public:
    void RunSimulationExample()
    {
        // 1. 初始化仿真系统
        simulationManager = gcnew SimulationManager();
        dataManager = gcnew ThreatSourceDataManager();
        simulationManager->StartSimulation(0.02);

        // 2. 创建目标
        auto target = gcnew BaseEquipment("target_001");
        target->KState = gcnew KinematicState();
        target->KState->Position = Vector3(5000, 0, 1000);

        // 3. 创建导弹
        auto missileData = dataManager->GetMissile("IR_Missile_Example");
        auto missile = gcnew InfraredImagingTerminalGuidanceMissile("missile_001", missileData);

        // 4. 注册实体
        simulationManager->RegisterEntity(target);
        simulationManager->RegisterEntity(missile);

        // 5. 运行仿真
        while (simulationTime < maxTime && missile->IsActive)
        {
            simulationManager->UpdateSimulation();
            // 处理仿真状态...
        }
    }
};

```

## 使用说明

### 基本仿真流程

#### 1. 初始化仿真管理器

```

var simulationManager = new SimulationManager();
var dataManager = new ThreatSourceDataManager();
simulationManager.StartSimulation(timeStep);

```

#### 2. 创建仿真实体

```

// 从配置创建导弹
var missileData = dataManager.GetMissile("missile_type");
var missile = new InfraredImagingTerminalGuidanceMissile("id", missileData);

// 创建目标
var target = new BaseEquipment("target_id");

```

#### 3. 配置实体状态

```
missile.KState = new KinematicState
{
    Position = new Vector3(x, y, z),
    Velocity = new Vector3(vx, vy, vz),
    Orientation = new Orientation(yaw, pitch, roll)
};
```

#### 4. 注册实体和事件

```
simulationManager.RegisterEntity(missile);
simulationManager.RegisterEntity(target);
simulationManager.Subscribe<MissileExplodeEvent>(OnMissileExplode);
```

#### 5. 运行仿真循环

```
while (condition)
{
    simulationManager.UpdateSimulation();
    // 处理仿真状态和结果
}
```

## 配置文件要求

示例代码需要相应的TOML配置文件：

```
# data/missiles/IR_Missile_Example.toml
Type = "missile"

[Name]
zh = "红外成像制导导弹示例"
en = "IR Imaging Guided Missile Example"

[Properties]
MaxSpeed = 800.0
MaxAcceleration = 50.0
MaxFlightTime = 120.0
MaxFlightDistance = 8000.0

[InfraredImagingGuidanceConfig]
MaxDetectionRange = 5000.0
FieldOfView = 30.0
TargetRecognitionProbability = 0.9
```

## C++/CLI项目配置

### 1. 创建C++/CLI项目

- 在Visual Studio中选择"CLR空项目"
- 设置目标框架为.NET 8.0

### 2. 添加引用

```
<Reference Include="ThreatSource">
  <HintPath>path\to\ThreatSource.dll</HintPath>
</Reference>
```

### 3. 编译设置

- 启用公共语言运行时支持 (/clr)
- 设置正确的.NET目标框架版本

## 注意事项

### 通用注意事项

1. **单位系统**: 确保所有参数使用正确的单位 (米、秒、弧度等)
2. **时间步长**: 合理设置仿真时间步长, 平衡精度和性能
3. **资源管理**: 正确处理仿真资源的创建和释放
4. **异常处理**: 添加适当的异常处理机制

## C#特定注意事项

1. 使用async/await处理异步操作
2. 正确订阅和取消订阅事件
3. 注意内存管理和垃圾回收

## C++/CLI特定注意事项

1. **平台限制**: 仅支持Windows平台
2. **内存管理**: 注意托管和非托管资源的正确释放
3. **类型转换**: 正确处理托管和原生类型之间的转换
4. **异常处理**: 使用托管异常处理机制

## 扩展示例

### 添加自定义事件处理

```
// 订阅多种事件
simulationManager.Subscribe<FlightPhaseChangeEvent>(OnFlightPhaseChange);
simulationManager.Subscribe<TargetHitEvent>(OnTargetHit);
simulationManager.Subscribe<GuidanceActivationEvent>(OnGuidanceActivation);

private void OnFlightPhaseChange(FlightPhaseChangeEvent evt)
{
    Console.WriteLine($"导弹 {evt.SenderId} 进入 {evt.NewPhase} 阶段");
}
```

### 多导弹仿真

```
// 创建多个导弹
for (int i = 0; i < missileCount; i++)
{
    var missile = new InfraredImagingTerminalGuidanceMissile($"missile_{i:D3}", missileData);
    // 设置不同的初始位置和参数
    simulationManager.RegisterEntity(missile);
}
```

### 实时状态监控

```
// 在仿真循环中监控状态
while (simulationManager.IsRunning)
{
    simulationManager.UpdateSimulation();

    // 获取所有导弹状态
    var missiles = simulationManager.GetEntitiesByType<BaseMissile>();
    foreach (var missile in missiles)
    {
        var status = missile.GetStatusInfo();
        // 处理状态信息
    }
}
```

# Class MissileProperties

导弹配置类，定义了导弹的所有基本属性和性能参数

## Inheritance

↳ [object](#)  
↳ MissileProperties

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MissileProperties
```

## Remarks

该类用于：

- 初始化导弹的基本参数
- 配置导弹的性能限制
- 设置导弹的物理特性
- 定义导弹的作战能力 所有导弹实例都基于此配置进行初始化

## Constructors

### MissileProperties()

初始化导弹配置类的新实例

## Declaration

```
public MissileProperties()
```

## Remarks

构造过程：

- 设置所有数值参数的默认值
- 初始化基本属性（ID、位置、朝向）
- 设置默认导弹类型

# Properties

## ExplosionRadius

获取或设置导弹的爆炸半径

Declaration

```
public double ExplosionRadius { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米 爆炸效果的有效作用范围 用于计算对目标的伤害

## HitProbability

获取或设置导弹的命中概率

Declaration

```
public double HitProbability { get; set; }
```

Property Value

Type	Description
double	

Remarks

范围：0-1 表示在理想条件下的命中可能性 用于评估导弹的作战效能

## Id

获取或设置导弹的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和追踪特定的导弹实例

## InitialOrientation

获取或设置导弹的初始朝向

Declaration

```
public Orientation InitialOrientation { get; set; }
```

Property Value

Type	Description
Orientation	

Remarks

使用欧拉角表示初始姿态 包含偏航角、俯仰角和滚转角

InitialPosition

获取或设置导弹的初始位置

Declaration

```
public Vector3D InitialPosition { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示发射位置 坐标系：右手坐标系，X向右，Y向上，Z向前

InitialSpeed

获取或设置导弹的初始速度

Declaration

```
public double InitialSpeed { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒 发射时的初始运动速度

LaunchAcceleration

获取或设置导弹的发射推力加速度

Declaration

```
public double LaunchAcceleration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒² 发射阶段的推进加速度 影响导弹的起飞性能

Mass

获取或设置导弹的质量

Declaration

```
public double Mass { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：千克 导弹的总质量，包括弹体和燃料 影响导弹的运动性能和惯性特性

## MaxAcceleration

获取或设置导弹的最大加速度

Declaration

```
public double MaxAcceleration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒<sup>2</sup> 导弹机动时的最大加速度限制 基于导弹的结构强度和推进系统能力

## MaxEngineBurnTime

获取或设置发动机的最大燃烧时间

Declaration

```
public double MaxEngineBurnTime { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 发动机能够持续工作的最长时间 基于燃料容量和燃烧速率

## MaxFlightDistance

获取或设置导弹的最大飞行距离

Declaration

```
public double MaxFlightDistance { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米 超过此距离导弹将自毁 基于导弹的燃料容量和设计射程

## MaxFlightTime



获取或设置导弹的最大飞行时间

Declaration

```
public double MaxFlightTime { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 超过此时间导弹将自毁 用于防止导弹无限飞行

MaxSpeed

获取或设置导弹的最大速度限制

Declaration

```
public double MaxSpeed { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒 导弹在飞行过程中不能超过此速度

ProportionalNavigationCoefficient

获取或设置导弹的比例导引系数

Declaration

```
public double ProportionalNavigationCoefficient { get; set; }
```

Property Value

Type	Description
double	

Remarks

无量纲参数 用于计算制导指令的增益 影响导弹的制导精度和稳定性

Type

获取或设置导弹的类型

Declaration

```
public MissileType Type { get; set; }
```

Property Value

Type	Description
MissileType	

Remarks

决定导弹的制导方式和行为特征 影响导弹的性能参数和作战能力

# Methods

## SetDefaultValues()

将所有数值参数设置为默认值

Declaration

```
public void SetDefaultValues()
```

Remarks

重置以下参数：

- 速度相关参数
- 时间和距离限制
- 加速度参数
- 性能参数 用于初始化或重置导弹配置

# Class LaserBeamUpdateEvent

激光波束更新事件，表示激光波束位置更新

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserBeamUpdateEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class LaserBeamUpdateEvent : SimulationEvent
```

## Remarks

用于实时更新激光波束的位置和方向 在制导过程中周期性触发

## Properties

### LaserBeamRiderId

获取或设置激光波束制导器的ID

#### Declaration

```
public string? LaserBeamRiderId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Class TerminalSensitiveMissile

末敏导弹类，实现了末端敏感引信的导弹功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [TerminalSensitiveMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Explode\(\)](#)  
[BaseMissile.Activate\(\)](#)  
[BaseMissile.Deactivate\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class TerminalSensitiveMissile : BaseMissile, IMissile
```

Remarks

该类提供了末敏导弹的完整实现：

- 继承自BaseMissile，具备基本的导弹功能
- 实现了末端敏感引信的特殊功能
- 支持子弹药分离和释放
- 具备高度和距离控制能力
- 提供精确的目标打击能力

工作流程：

1. 导弹发射并飞向预定分离点
2. 到达分离高度时释放子弹药
3. 子弹药独立制导飞向目标
4. 母弹完成任务后自动销毁

Constructors

TerminalSensitiveMissile(string, MissileProperties, ISimulationManager)

初始化末敏导弹的新实例

Declaration

```
public TerminalSensitiveMissile(string targetId, MissileProperties properties, ISimulationManager manager)
```

Parameters

Type	Name	Description
string	targetId	目标的唯一标识符
MissileProperties	properties	导弹的配置参数
ISimulationManager	manager	仿真管理器实例

Remarks

构造过程：

1. 初始化基本属性
2. 创建子弹药数组
3. 计算分离点位置
4. 计算最佳发射角度
5. 设置初始速度和方向

Exceptions

Type	Condition
Exception	当目标不存在时抛出
InvalidOperationException	当无法计算发射方向时抛出

Methods

GetStatus()

获取导弹的状态信息字符串

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基类状态信息
- 分离点位置
- 母弹特有的状态信息

## Update(double)

更新导弹的状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

1. 检查导弹是否处于活动状态
2. 验证自毁条件（时间、距离、高度）
3. 更新基本运动状态
4. 计算与分离点的距离
5. 记录状态信息
6. 必要时触发分离动作

自毁条件：

- 超出最大飞行时间
- 超出最大飞行距离
- 高度低于地面

## Implements

[IMissile](#)

# Class InfraredGuidanceCommandEvent

红外指令制导事件，表示发送制导指令

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ InfraredGuidanceCommandEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredGuidanceCommandEvent : SimulationEvent
```

## Remarks

用于红外指令制导系统 包含目标和导弹的视线向量信息

## Properties

### TargetMissileId

获取或设置目标导弹的ID

#### Declaration

```
public string? TargetMissileId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识接收制导指令的导弹

## TrackerToMissileVector

获取或设置跟踪器到导弹的视线向量

Declaration

```
public Vector3D TrackerToMissileVector { get; set; }
```

Property Value

Type	Description
<a href="#">Vector3D</a>	

Remarks

表示从跟踪器到导弹的方向 单位：米

## TrackerToTargetVector

获取或设置跟踪器到目标的视线向量

Declaration

```
public Vector3D TrackerToTargetVector { get; set; }
```

Property Value

Type	Description
<a href="#">Vector3D</a>	

Remarks

表示从跟踪器到目标的方向 单位：米



# Class InfraredDetector

红外探测器类，实现了红外辐射的探测和目标识别功能

## Inheritance

↳ [object](#)  
↳ [Sensor](#)  
↳ [InfraredDetector](#)

## Implements

[ISensor](#)

## Inherited Members

[Sensor.IsActive](#)  
[Sensor.Position](#)  
[Sensor.Orientation](#)  
[Sensor.Activate\(\)](#)  
[Sensor.Deactivate\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredDetector : Sensor, ISensor
```

## Remarks

该类提供了红外探测器的核心功能：

- 红外辐射强度测量
- 目标存在性判断
- 视场角范围探测
- 实时状态更新 用于末敏子弹的目标探测和跟踪

## Constructors

### **InfraredDetector(TerminalSensitiveSubmunition, double, double)**

初始化红外探测器的新实例

## Declaration

```
public InfraredDetector(TerminalSensitiveSubmunition submunition, double detectionRange, double fieldOfView)
```

Parameters

Type	Name	Description
TerminalSensitiveSubmunition	<i>submunition</i>	未敏子弹实例
double	<i>detectionRange</i>	探测范围，单位：米
double	<i>fieldOfView</i>	视场角，单位：度

Remarks

构造过程：

- 设置探测参数
- 关联未敏子弹
- 初始化传感器数据
- 继承基类位置和姿态

## Properties

### DetectionRange

获取或设置探测范围，单位：米

Declaration

```
public double DetectionRange { get; set; }
```

Property Value

Type	Description
double	

Remarks

定义了探测器的最大作用距离 影响目标探测的有效范围

### FieldOfView

获取或设置视场角，单位：度

Declaration

```
public double FieldOfView { get; set; }
```

Property Value

Type	Description
double	

Remarks

定义了探测器的视场范围 影响目标探测的空间覆盖

## Methods

### GetSensorData()

获取红外探测器的最新数据

Declaration

```
public override SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含探测结果的数据对象

Overrides

[Sensor.GetSensorData\(\)](#)

Remarks

返回数据：

- 目标探测状态
- 温度测量结果
- 时间戳信息

## Update(double)

更新红外探测器的工作状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Overrides

[Sensor.Update\(double\)](#)

Remarks

更新过程：

- 检查激活状态
- 测量辐射强度
- 判断目标存在
- 更新传感器数据

## Implements

[ISensor](#)

# Namespace ThreatSource.Guidance

## Classes

### **BasicGuidanceSystem**

基础制导系统类，实现了制导系统的基本功能框架

### **InfraredCommandGuidanceSystem**

红外指令导引系统类，实现了基于红外跟踪器的指令制导功能

### **InfraredImagingGuidanceSystem**

红外成像引导系统类，实现了基于红外图像的目标探测和跟踪功能

### **LaserBeamRiderGuidanceSystem**

激光驾束制导系统类，实现了基于激光束跟踪的制导功能

### **LaserSemiActiveGuidanceSystem**

激光半主动制导系统类，实现了基于激光照射的目标跟踪和制导功能

### **MillimeterWaveGuidanceSystem**

毫米波导引头系统类，实现了基于毫米波雷达的目标探测和跟踪功能

## Interfaces

### **IGuidanceSystem**

制导系统接口，定义了所有制导系统的通用功能

[Show / Hide Table of Contents](#)

# Interface ISimulationManager

仿真管理器接口，提供仿真系统的核心功能

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface ISimulationManager
```

## Remarks

该接口定义了仿真系统的主要功能，包括：

- 事件系统：用于实体间的通信和状态同步
- 实体管理：负责实体的注册、注销和查询
- 第三方集成：支持与其他仿真环境的对接

## Methods

### GetAllEntities()

获取仿真系统中的所有实体

Declaration

```
IReadOnlyList<object> GetAllEntities()
```

Returns

Type	Description
<a href="#">IReadOnlyList&lt;object&gt;</a>	所有实体的列表

### GetEntitiesByType<T>()

获取特定类型的所有实体

Declaration

```
IReadOnlyList<T> GetEntitiesByType<T>() where T : class
```

Returns

Type	Description
<a href="#">IReadOnlyList&lt;T&gt;</a>	指定类型的实体列表

Type Parameters

Name	Description
<i>T</i>	实体类型

## GetEntityById(string)

根据ID获取实体

Declaration

```
object? GetEntityById(string id)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	实体ID

Returns

Type	Description
<a href="#">object</a>	实体对象，如果不存在则返回null

## GetSimulationAdapter()

获取当前的仿真环境适配器

Declaration

```
ISimulationAdapter? GetSimulationAdapter()
```

Returns

Type	Description
<a href="#">ISimulationAdapter</a>	当前配置的适配器实例，如果未配置则返回null

## PublishEvent<T>(T)

发布事件到仿真系统

Declaration

```
void PublishEvent<T>(T evt)
```

Parameters

Type	Name	Description
<i>T</i>	<i>evt</i>	要发布的事件

Type Parameters

Name	Description
<i>T</i>	事件类型

## RegisterEntity(string, object)

注册实体到仿真系统

Declaration

```
bool RegisterEntity(string id, object entity)
```

#### Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	实体ID
<a href="#">object</a>	<i>entity</i>	实体对象

#### Returns

Type	Description
<a href="#">bool</a>	注册是否成功

## SetSimulationAdapter(ISimulationAdapter)

设置第三方仿真环境适配器

#### Declaration

```
void SetSimulationAdapter(ISimulationAdapter adapter)
```

#### Parameters

Type	Name	Description
<a href="#">ISimulationAdapter</a>	<i>adapter</i>	要设置的适配器实例

#### Remarks

用于配置与外部仿真环境的集成

## SubscribeToEvent<T>(Action<T>)

订阅特定类型的事件

#### Declaration

```
void SubscribeToEvent<T>(Action<T> handler)
```

#### Parameters

Type	Name	Description
<a href="#">Action&lt;T&gt;</a>	<i>handler</i>	事件处理函数

#### Type Parameters

Name	Description
<i>T</i>	事件类型

## UnregisterEntity(string)

从仿真系统注销实体

#### Declaration

```
bool UnregisterEntity(string id)
```

#### Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	要注销的实体ID

Returns

Type	Description
bool	注销是否成功

UnsubscribeFromEvent<T>(Action<T>)

取消订阅特定类型的事件

Declaration

```
void UnsubscribeFromEvent<T>(Action<T> handler)
```

Parameters

Type	Name	Description
Action<T>	handler	要取消的事件处理函数

Type Parameters

Name	Description
T	事件类型



[Show / Hide Table of Contents](#)

# Class InfraredWarnerAlarmStopEvent

红外告警器警报停止事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [InfraredWarnerAlarmStopEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class InfraredWarnerAlarmStopEvent : SimulationEvent
```

## Properties

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Class SensorData

传感器数据的抽象基类，定义了所有传感器数据的通用属性

## Inheritance

↳ [object](#)

- ↳ [SensorData](#)
  - ↳ [AltimeterSensorData](#)
  - ↳ [InfraredSensorData](#)
  - ↳ [RadiometerSensorData](#)
  - ↳ [RangefinderSensorData](#)

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public abstract class SensorData
```

## Remarks

该类作为所有传感器数据类型的基础：

- 提供基本的时间戳信息
- 统一数据接口规范
- 支持数据类型扩展 用于传感器数据的采集和处理

# Properties

## Timestamp

获取或设置数据采集的时间戳

## Declaration

```
public DateTime Timestamp { get; set; }
```

## Property Value

Type	Description
<a href="#">DateTime</a>	

Remarks

记录传感器数据的采集时间 用于数据时序分析和同步处理

[Show / Hide Table of Contents](#)

# Class InfraredTrackerConfig

红外测角仪配置类

## Inheritance

↳ [object](#)  
↳ InfraredTrackerConfig

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredTrackerConfig
```

## Constructors

### InfraredTrackerConfig()

构造函数,设置默认值

#### Declaration

```
public InfraredTrackerConfig()
```

## Properties

### AngleMeasurementAccuracy

角度测量精度(弧度)

#### Declaration

```
public double AngleMeasurementAccuracy { get; set; }
```

#### Property Value

Type	Description
double	

## FieldOfView

视场角(弧度)

Declaration

```
public double FieldOfView { get; set; }
```

Property Value

Type	Description
double	

## Id

红外测角仪ID

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

## InitialOrientation

初始朝向

Declaration

```
public Orientation InitialOrientation { get; set; }
```

Property Value

Type	Description
Orientation	

## InitialPosition

初始位置

Declaration

```
public Vector3D InitialPosition { get; set; }
```

Property Value

Type	Description
Vector3D	

## MaxTrackingRange

最大跟踪距离(米)

Declaration

```
public double MaxTrackingRange { get; set; }
```

Property Value

Type	Description
double	

UpdateFrequency

更新频率(赫兹)

Declaration

```
public double UpdateFrequency { get; set; }
```

Property Value

Type	Description
double	

[Show / Hide Table of Contents](#)

# Class LaserJammerConfig

激光干扰器配置类

## Inheritance

↳ [object](#)  
↳ LaserJammerConfig

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserJammerConfig
```

## Constructors

### LaserJammerConfig()

构造函数,设置默认值

#### Declaration

```
public LaserJammerConfig()
```

## Properties

### Id

激光干扰器ID

#### Declaration

```
public string Id { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

## InitialJammingPower

初始干扰功率(瓦特)

Declaration

```
public double InitialJammingPower { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## MaxJammingCooldown

最大干扰冷却时间(秒)

Declaration

```
public double MaxJammingCooldown { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## MaxJammingPower

最大干扰功率(瓦特)

Declaration

```
public double MaxJammingPower { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## PowerIncreaseRate

功率增加速率(瓦特/秒)

Declaration

```
public double PowerIncreaseRate { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	



# Class MillimeterWaveTerminalGuidedMissile

毫米波末制导导弹类，继承自基础导弹类

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [MillimeterWaveTerminalGuidedMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Activate\(\)](#)  
[BaseMissile.Deactivate\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class MillimeterWaveTerminalGuidedMissile : BaseMissile, IMissile
```

### Remarks

该类提供了毫米波末制导导弹的完整实现：

- 继承自BaseMissile，具备基本的导弹功能
- 实现了毫米波制导系统
- 支持多阶段飞行控制
- 具备全天候目标探测能力
- 提供精确的末制导打击能力

工作流程：

1. 导弹发射并进入发射阶段
2. 切换到巡航阶段，进行中程飞行
3. 进入末制导阶段，启动毫米波制导
4. 接近目标后引爆或达到自毁条件后自毁

## Constructors

### MillimeterWaveTerminalGuidedMissile(MissileProperties, ISimulationManager)

初始化毫米波末制导导弹的新实例

Declaration

```
public MillimeterWaveTerminalGuidedMissile(MissileProperties properties, ISimulationManager manager)
```

Parameters

Type	Name	Description
<a href="#">MissileProperties</a>	<i>properties</i>	导弹的配置参数
<a href="#">ISimulationManager</a>	<i>manager</i>	仿真管理器实例

Remarks

构造过程：

1. 调用基类构造函数
2. 创建制导系统实例
3. 设置初始飞行阶段

## Methods

### Explode()

执行导弹爆炸

Declaration

```
public override void Explode()
```

Overrides

[BaseMissile.Explode\(\)](#)

Remarks

爆炸过程：

1. 调用基类的爆炸方法
2. 切换到爆炸阶段

### GetStatus()

获取导弹的状态信息字符串

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基类状态信息
- 当前飞行阶段
- 制导系统状态
- 目标跟踪信息

### Update(double)

更新导弹的状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

1. 调用基类的更新方法
2. 根据当前阶段选择更新方法
3. 执行相应阶段的更新逻辑
4. 检查是否需要自毁

### Implements

[IMissile](#)

# Class TerminalSensitiveSubmunition

末敏子弹类，实现了多传感器融合的末端制导功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [TerminalSensitiveSubmunition](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Explode\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class TerminalSensitiveSubmunition : BaseMissile, IMissile
```

Remarks

该类提供了末敏子弹的核心功能：

- 多传感器融合（红外、毫米波、激光）
- 螺旋扫描搜索
- 自主目标识别
- 末端精确制导 通过多种传感器的协同工作实现对目标的精确打击

Constructors

TerminalSensitiveSubmunition(string, MissileProperties, ISimulationManager)

初始化末敏子弹的新实例

Declaration

```
public TerminalSensitiveSubmunition(string targetId, MissileProperties properties, ISimulationManager manager)
```

Parameters

Type	Name	Description
string	targetId	目标ID
MissileProperties	properties	子弹配置参数
ISimulationManager	manager	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性
- 创建传感器系统
- 配置传感器参数
- 设置初始状态

Methods

Activate()

激活子弹

Declaration

```
public override void Activate()
```

Overrides

BaseMissile.Activate()

Remarks

激活过程：

- 调用基类激活方法
- 订阅目标辐射事件
- 准备传感器系统

Deactivate()

停用子弹

Declaration

```
public override void Deactivate()
```

Overrides

[BaseMissile.Deactivate\(\)](#)

Remarks

停用过程：

- 调用基类停用方法
- 停用所有传感器
- 清理制导状态

DetectTarget(double)

检测目标是否在视场内

Declaration

```
public bool DetectTarget(double fieldOfView)
```

Parameters

Type	Name	Description
double	fieldOfView	视场角，单位：度

Returns

Type	Description
bool	如果目标在视场内返回true，否则返回false

Remarks

检测过程：

- 获取目标位置
- 计算目标方向
- 判断是否在视场内

GetStatus()

获取子弹状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含子弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基本状态信息
- 当前飞行阶段
- 扫描和检测状态
- 传感器工作状态

## Update(double)

更新子弹状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	deltaTime	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

- 更新所有传感器
- 根据当前阶段更新状态
- 处理阶段转换
- 调用基类更新

## Implements

[IMissile](#)

# Class LaserBeamRider

激光波束制导器类，用于生成激光波束制导场

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [LaserBeamRider](#)

## Implements

[IIndicator](#)

## Inherited Members

[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserBeamRider : SimulationElement, IIndicator
```

## Remarks

该类提供了激光波束制导系统的核心功能：

- 生成和维护激光波束制导场
- 控制激光波束的开启和关闭
- 实时更新波束位置和方向
- 发布波束状态事件

## Constructors

**LaserBeamRider(string, string, string, double, LaserBeamRiderConfig, ISimulationManager)**

初始化激光波束制导器的新实例



Declaration

```
public LaserBeamRider(string id, string missileId, string targetId, double speed, LaserBeamRiderConfig config, ISimulationManager simulationManager)
```

Parameters

Type	Name	Description
string	id	制导器的唯一标识符
string	missileId	导弹ID
string	targetId	目标ID
double	speed	初始速度
LaserBeamRiderConfig	config	配置参数
ISimulationManager	simulationManager	仿真管理器实例

Remarks

- 构造过程：
- 初始化基本属性
  - 设置激光参数
  - 配置制导范围
  - 建立目标关联

## Properties

### BeamDivergence

获取或设置激光发散角

Declaration

```
public double BeamDivergence { get; }
```

Property Value

Type	Description
double	

Remarks

单位：毫弧度 影响激光波束的扩散特性和制导精度

### ControlFieldDiameter

获取或设置控制场直径

Declaration

```
public double ControlFieldDiameter { get; }
```

Property Value

Type	Description
double	

Remarks

单位：米 定义了激光波束制导场的有效范围

## IsBeamOn

获取激光束是否开启

Declaration

```
public bool IsBeamOn { get; }
```

Property Value

Type	Description
bool	

Remarks

true表示激光波束正在发射 false表示激光波束已关闭

## LaserDirection

获取或设置激光方向

Declaration

```
public Vector3D LaserDirection { get; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示激光波束的指向 向量的模为1（归一化）

## LaserPower

获取或设置激光功率

Declaration

```
public double LaserPower { get; }
```

Property Value

Type	Description
double	

Remarks

单位： 瓦特 影响激光波束的有效作用距离和制导精度

## MaxGuidanceDistance

获取或设置最大导引距离

Declaration

```
public double MaxGuidanceDistance { get; }
```

Property Value

Type	Description
double	

Remarks

单位：米 超出此距离的导弹将无法接收到有效的制导信号

### MissileId

获取当前制导的导弹ID

Declaration

```
public string? MissileId { get; }
```

Property Value

Type	Description
string	

Remarks

记录正在接受制导的导弹标识符 如果为null表示当前没有制导目标

### TargetId

获取目标ID

Declaration

```
public string? TargetId { get; }
```

Property Value

Type	Description
string	

Remarks

记录当前跟踪的目标标识符 如果为null表示当前没有跟踪目标

## Methods

### Activate()

激活激光驾束仪

Declaration

```
public override void Activate()
```

Overrides

[SimulationElement.Activate\(\)](#)

Remarks

激活过程：

- 设置激活状态
- 开始激光照射
- 发布激活事件

### Deactivate()

停用激光驾束仪

Declaration

```
public override void Deactivate()
```

Overrides

[SimulationElement.Deactivate\(\)](#)

Remarks

停用过程：

- 停止激光照射
- 清理状态
- 发布停用事件

**GetRunningState()**

获取激光驾束仪运行状态

Declaration

```
public IndicatorRunningState GetRunningState()
```

Returns

Type	Description
<a href="#">IndicatorRunningState</a>	包含完整状态信息的结构体

Remarks

返回信息包括：

- 目标和导弹ID
- 设备类型和工作状态
- 位置和朝向信息
- 激活状态

**GetStatus()**

获取激光驾束仪状态字符串

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含设备状态的详细描述

Overrides

[SimulationElement.GetStatus\(\)](#)

Remarks

返回信息包括：

- 设备ID和位置
- 激光参数（功率、发散角等）
- 工作状态（激活、照射等）
- 制导范围参数

**StartBeamIllumination()**

开启激光束照射

Declaration

```
public void StartBeamIllumination()
```

#### Remarks

开启过程：

- 计算初始指向
- 设置照射状态
- 发布开始事件

## StopBeamIllumination()

停止激光束照射

#### Declaration

```
public void StopBeamIllumination()
```

#### Remarks

停止过程：

- 关闭激光输出
- 清除指向信息
- 发布停止事件

## Update(double)

更新激光驾束仪状态

#### Declaration

```
public override void Update(double deltaTime)
```

#### Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

#### Overrides

[SimulationElement.Update\(double\)](#)

#### Remarks

更新过程：

- 检查激活状态
- 更新激光指向
- 发布状态更新事件

## Implements

[IIndicator](#)

# Class InfraredDetectionEvent

红外探测事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [InfraredDetectionEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class InfraredDetectionEvent : SimulationEvent
```

## Properties

### Intensity

红外辐射强度

#### Declaration

```
public double Intensity { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

威胁源仿真库文档

[Back to top](#)

# Interface ITarget

定义目标对象的基本接口，提供目标的基本特征和属性

**Namespace:** [ThreatSource.Target](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface ITarget
```

## Remarks

该接口定义了目标的关键特征：

- 目标类型标识
- 雷达散射截面积
- 红外辐射特征 用于在仿真系统中表示和识别不同类型的目标

## Properties

### InfraredRadiationIntensity

获取目标的红外辐射强度

Declaration

```
double InfraredRadiationIntensity { get; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：瓦特/平方米 表示目标的热辐射特征 影响红外制导系统的探测和跟踪能力

### RadarCrossSection

获取目标的雷达散射截面积

Declaration

```
double RadarCrossSection { get; }
```

Property Value

Type	Description
<a href="#">double</a>	



Remarks

单位：平方米 表示目标对雷达波的反射能力 影响目标在雷达系统中的探测距离和识别概率

Type

获取目标的类型标识

Declaration

```
string Type { get; }
```

Property Value

Type	Description
string	

Remarks

用于区分不同种类的目标，如坦克、装甲车等 在仿真系统中用于目标识别和分类

# Class InfraredCommandGuidedMissile

红外指令制导导弹类，实现了红外指令制导的导弹功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [InfraredCommandGuidedMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.Update\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Explode\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class InfraredCommandGuidedMissile : BaseMissile, IMissile
```

Remarks

该类提供了红外指令制导导弹的核心功能：

- 红外热源管理（点亮和熄灭）
- 指令制导控制
- 飞行阶段管理
- 状态监控和事件处理 通过红外测角仪和指令制导系统实现对导弹的制导控制

Constructors

InfraredCommandGuidedMissile(MissileProperties, ISimulationManager)

初始化红外指令制导导弹的新实例

Declaration

```
public InfraredCommandGuidedMissile(MissileProperties config, ISimulationManager manager)
```

Parameters

Type	Name	Description
MissileProperties	config	导弹配置参数
ISimulationManager	manager	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性
- 设置红外辐射功率
- 创建指令导引系统
- 配置PID参数

Properties

RadiationPower

获取或设置红外热源辐射功率

Declaration

```
public double RadiationPower { get; set; }
```

Property Value

Type	Description
double	辐射功率，单位：瓦特

Remarks

影响导弹的红外信号强度 用于红外测角仪的跟踪和制导

Methods

Activate()

激活导弹

Declaration

```
public override void Activate()
```

Overrides

[BaseMissile.Activate\(\)](#)

Remarks

- 激活过程：
- 调用基类激活方法
  - 订阅制导指令事件
  - 准备接收制导指令

Deactivate()

停用导弹

Declaration

```
public override void Deactivate()
```

Overrides

[BaseMissile.Deactivate\(\)](#)

Remarks

- 停用过程：
- 调用基类停用方法
  - 熄灭红外热源
  - 取消订阅制导事件

GetStatus()

获取导弹状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

- 返回信息包括：
- 基本状态信息
  - 当前飞行阶段
  - 制导系统状态

LightInfraredSource()

点亮红外热源

Declaration

```
public void LightInfraredSource()
```

Remarks  
发布红外热源点亮事件 包含当前辐射功率信息

### LightOffInfraredSource()

熄灭红外热源

Declaration

```
public void LightOffInfraredSource()
```

Remarks  
发布红外热源熄灭事件 停止红外信号发射

### UpdateMotionState(double)

更新导弹运动状态

Declaration

```
protected override void UpdateMotionState(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides  
[BaseMissile.UpdateMotionState\(double\)](#)

Remarks  
更新过程：

- 点亮红外热源
- 根据当前阶段更新状态
- 调用基类的运动更新

### Implements

[IMissile](#)

- [ThreatSource.Guidance](#)
  - [BasicGuidanceSystem](#)
  - [IGuidanceSystem](#)
  - [InfraredCommandGuidanceSystem](#)
  - [InfraredImagingGuidanceSystem](#)
  - [LaserBeamRiderGuidanceSystem](#)
  - [LaserSemiActiveGuidanceSystem](#)
  - [MillimeterWaveGuidanceSystem](#)
- [ThreatSource.Indicator](#)
  - [Indicator](#)
  - [IndicatorRunningState](#)
  - [IndicatorState](#)
  - [IndicatorType](#)
  - [InfraredTracker](#)
  - [LaserBeamRider](#)
  - [LaserDesignator](#)
- [ThreatSource.Missile](#)
  - [BaseMissile](#)
  - [IMissile](#)
  - [InfraredCommandGuidedMissile](#)
  - [InfraredImagingTerminalGuidedMissile](#)
  - [LaserBeamRiderMissile](#)
  - [LaserSemiActiveGuidedMissile](#)
  - [MillimeterWaveTerminalGuidedMissile](#)
  - [MissileProperties](#)
  - [MissileRunningState](#)
  - [MissileType](#)
  - [TerminalSensitiveMissile](#)
  - [TerminalSensitiveSubmunition](#)
- [ThreatSource.Sensor](#)
  - [AltimeterSensorData](#)
  - [ISensor](#)
  - [InfraredDetector](#)
  - [InfraredSensorData](#)
  - [LaserRangefinder](#)
  - [MillimeterWaveAltimeter](#)
  - [MillimeterWaveRadiometer](#)
  - [RadiometerSensorData](#)
  - [RangefinderSensorData](#)
  - [Sensor](#)
  - [SensorData](#)
- [ThreatSource.Simulation](#)
  - [EntityActivatedEvent](#)
  - [EntityDeactivatedEvent](#)
  - [EntityDestroyedEvent](#)
  - [ISimulationAdapter](#)
  - [ISimulationManager](#)
  - [InfraredDetectionEvent](#)
  - [InfraredGuidanceCommandEvent](#)
  - [InfraredGuidanceMissileLightEvent](#)
  - [InfraredGuidanceMissileLightOffEvent](#)
  - [InfraredJammingEvent](#)
  - [InfraredTrackerConfig](#)
  - [InfraredWarnerAlarmEvent](#)
  - [InfraredWarnerAlarmStopEvent](#)
  - [InfraredWarnerConfig](#)
  - [LaserBeamRiderConfig](#)
  - [LaserBeamStartEvent](#)
  - [LaserBeamStopEvent](#)
  - [LaserBeamUpdateEvent](#)
  - [LaserDesignatorConfig](#)
  - [LaserIlluminationStartEvent](#)
  - [LaserIlluminationStopEvent](#)
  - [LaserIlluminationUpdateEvent](#)
  - [LaserJammerConfig](#)
  - [LaserJammingEvent](#)

- [LaserWarnerAlarmEvent](#)
- [LaserWarnerAlarmStopEvent](#)
- [LaserWarnerConfig](#)
- [MillimeterWaveDetectionEvent](#)
- [MillimeterWaveJammerConfig](#)
- [MillimeterWaveJammingEvent](#)
- [MillimeterWaveWarnerAlarmEvent](#)
- [MillimeterWaveWarnerAlarmStopEvent](#)
- [MillimeterWaveWarnerConfig](#)
- [MissileFireEvent](#)
- [SimulationElement](#)
- [SimulationEvent](#)
- [SimulationManager](#)
- [TankRadiationEvent](#)
- [TargetDestroyedEvent](#)
- [TargetHitEvent](#)
- [UltravioletDetectionEvent](#)
- [UltravioletWarnerAlarmEvent](#)
- [UltravioletWarnerAlarmStopEvent](#)
- [UltravioletWarnerConfig](#)
- [ThreatSource.Simulation.Testing](#)
  - [TestSimulationAdapter](#)
- [ThreatSource.Target](#)
  - [ITarget](#)
  - [Tank](#)
- [ThreatSource.Utils](#)
  - [MotionAlgorithm](#)
  - [Orientation](#)
  - [Vector2D](#)
  - [Vector3D](#)

# Class LaserDesignator

激光指示器类，实现了对目标的激光照射和制导功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [LaserDesignator](#)

## Inherited Members

[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserDesignator : SimulationElement
```

## Remarks

该类提供了激光指示器的核心功能：

- 激光目标照射
- 抗干扰处理
- 事件管理
- 状态监控 通过激光照射目标实现半主动激光制导

## Constructors

### **LaserDesignator(string, string, string, double, LaserDesignatorConfig, ISimulationManager)**

初始化激光指示器的新实例

## Declaration



```
public LaserDesignator(string id, string targetId, string missileId, double speed, LaserDesignatorConfig config, ISimulationManager simulationManager)
```

Parameters

Type	Name	Description
string	<i>id</i>	激光指示器ID
string	<i>targetId</i>	目标ID
string	<i>missileId</i>	导弹ID
double	<i>speed</i>	移动速度
LaserDesignatorConfig	<i>config</i>	配置参数
ISimulationManager	<i>simulationManager</i>	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性
- 设置目标和导弹关联
- 配置激光参数
- 设置初始状态

## Properties

### IsIlluminationOn

获取或设置是否正在照射状态

Declaration

```
public bool IsIlluminationOn { get; }
```

Property Value

Type	Description
bool	

Remarks

true表示正在执行激光照射 false表示未在照射状态 控制激光发射状态

### IsJammed

获取或设置是否被干扰状态

Declaration

```
public bool IsJammed { get; }
```

Property Value

Type	Description
bool	

Remarks

true表示当前受到有效干扰 false表示工作正常 影响激光照射的有效性

### JammingThreshold

获取或设置干扰阈值，单位：分贝

Declaration

```
public double JammingThreshold { get; }
```

Property Value

Type	Description
double	

Remarks

定义了激光指示器的抗干扰能力 当干扰信号强度超过此阈值时触发干扰效果

LaserDivergenceAngle

获取或设置激光发散角，单位：弧度

Declaration

```
public double LaserDivergenceAngle { get; }
```

Property Value

Type	Description
double	

Remarks

定义了激光束的扩展角度 影响照射面积和能量密度

LaserPower

获取或设置激光功率，单位：瓦特

Declaration

```
public double LaserPower { get; }
```

Property Value

Type	Description
double	

Remarks

定义了激光照射的能量强度 影响照射距离和制导效果

MissileId

获取或设置导弹ID

Declaration

```
public string? MissileId { get; }
```

Property Value

Type	Description
string	

Remarks

记录当前关联的导弹标识符 用于导弹制导控制

## TargetId

获取或设置目标ID

Declaration

```
public string? TargetId { get; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

记录当前照射的目标标识符 用于目标识别和状态更新

## Methods

### Activate()

激活激光指示器

Declaration

```
public override void Activate()
```

Overrides

[SimulationElement.Activate\(\)](#)

Remarks

激活过程：

- 设置激活状态
- 清除干扰状态
- 开始激光照射
- 订阅相关事件
- 调用基类激活

### Deactivate()

停用激光指示器

Declaration

```
public override void Deactivate()
```

Overrides

[SimulationElement.Deactivate\(\)](#)

Remarks

停用过程：

- 清除激活状态
- 停止激光照射
- 取消事件订阅
- 清理工作状态
- 调用基类停用

### GetRunningState()

获取激光指示器运行状态

Declaration

```
public IndicatorRunningState GetRunningState()
```

Returns

Type	Description
<a href="#">IndicatorRunningState</a>	包含指示器完整状态信息的结构体

Remarks

返回信息包括：

- 目标和导弹关联状态
- 工作类型和状态
- 位置和姿态信息
- 激活状态

GetStatus()

获取激光指示器状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含指示器状态的详细描述

Overrides

[SimulationElement.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基本标识信息
- 位置信息
- 目标关联状态
- 工作状态
- 干扰状态
- 激光参数

Update(double)

更新激光指示器状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[SimulationElement.Update\(double\)](#)

Remarks

更新过程：

- 检查激活状态
- 检查干扰状态

- 更新照射状态
- 发布状态事件

[Show / Hide Table of Contents](#)

# Class MillimeterWaveJammerConfig

毫米波干扰器配置类

## Inheritance

↳ [object](#)

↳ MillimeterWaveJammerConfig

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveJammerConfig
```

## Constructors

### MillimeterWaveJammerConfig()

构造函数

#### Declaration

```
public MillimeterWaveJammerConfig()
```

## Properties

### Id

干扰器ID

#### Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

## InitialJammingPower

初始干扰功率(瓦特)

Declaration

```
public double InitialJammingPower { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## MaxJammingCooldown

最大冷却时间(秒)

Declaration

```
public double MaxJammingCooldown { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## MaxJammingPower

最大干扰功率(瓦特)

Declaration

```
public double MaxJammingPower { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## PowerIncreaseRate

功率增长率(瓦特/秒)

Declaration

```
public double PowerIncreaseRate { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

[Show / Hide Table of Contents](#)

# Class InfraredGuidanceMissileLightOffEvent

红外指令制导导弹熄灭红外热源事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ InfraredGuidanceMissileLightOffEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredGuidanceMissileLightOffEvent : SimulationEvent
```

## Remarks

用于模拟导弹尾部红外热源的关闭 触发时机：导弹发动机关闭时



# Class BaseMissile

导弹基类，实现了导弹的基本功能和状态管理

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [InfraredCommandGuidedMissile](#)  
↳ [InfraredImagingTerminalGuidedMissile](#)  
↳ [LaserBeamRiderMissile](#)  
↳ [LaserSemiActiveGuidedMissile](#)  
↳ [MillimeterWaveTerminalGuidedMissile](#)  
↳ [TerminalSensitiveMissile](#)  
↳ [TerminalSensitiveSubmunition](#)

## Implements

[IMissile](#)

## Inherited Members

[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class BaseMissile : SimulationElement, IMissile
```

## Remarks

该类提供了导弹的核心功能：

- 运动状态计算和更新
- 制导系统管理
- 发动机控制
- 自毁和爆炸处理
- 状态监控和事件处理 所有具体的导弹类型都继承自此基类

# Constructors

## BaseMissile(MissileProperties, ISimulationManager)

初始化导弹基类的新实例

Declaration

```
public BaseMissile(MissileProperties properties, ISimulationManager manager)
```

Parameters

Type	Name	Description
<a href="#">MissileProperties</a>	<i>properties</i>	导弹的配置参数
<a href="#">ISimulationManager</a>	<i>manager</i>	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性（位置、朝向、速度）
- 设置初始状态（未激活、未制导）
- 清零计时器和计数器
- 初始化加速度向量

# Fields

## LastKnownVelocity

获取或设置导弹的最后已知速度向量

Declaration

```
protected Vector3D LastKnownVelocity
```

Field Value

Type	Description
<a href="#">Vector3D</a>	三维速度向量

Remarks

用于在失去制导时保持导弹的运动状态 作为弹道计算的参考数据

## MissileProperties

获取导弹的固定配置参数

Declaration

```
public readonly MissileProperties MissileProperties
```

Field Value

Type	Description
<a href="#">MissileProperties</a>	导弹属性配置对象

Remarks

包含导弹的所有基本属性和性能限制 在导弹创建时设置，运行期间保持不变

# Properties

## EngineBurnTime

获取发动机的当前燃烧时间

Declaration

```
public double EngineBurnTime { get; protected set; }
```

Property Value

Type	Description
double	发动机燃烧时间，单位：秒

Remarks

发动机工作的累计时间 用于控制推力变化和燃料消耗

## FlightDistance

获取导弹的当前飞行距离

Declaration

```
public double FlightDistance { get; protected set; }
```

Property Value

Type	Description
double	飞行距离，单位：米

Remarks

从发射点到当前位置的累计飞行距离 用于判断是否超出最大射程

## FlightTime

获取导弹的当前飞行时间

Declaration

```
public double FlightTime { get; protected set; }
```

Property Value

Type	Description
double	飞行时间，单位：秒

Remarks

从发射时刻开始计时 用于控制导弹的生命周期

## GuidanceAcceleration

获取或设置导弹的制导加速度

Declaration

```
protected Vector3D GuidanceAcceleration { get; set; }
```

Property Value

Type	Description
Vector3D	三维加速度向量，单位：米/秒 <sup>2</sup>

Remarks

由制导系统计算得出的期望加速度 用于修正导弹的飞行路径

IsGuidance

获取导弹是否处于制导状态

Declaration

```
public bool IsGuidance { get; protected set; }
```

Property Value

Type	Description
bool	true表示导弹当前在制导，false表示导弹处于非制导状态

Remarks

影响导弹的运动学计算方法 制导状态下使用制导律计算加速度 非制导状态下使用弹道方程计算运动

LostGuidanceTime

获取或设置导弹失去制导的持续时间

Declaration

```
protected double LostGuidanceTime { get; set; }
```

Property Value

Type	Description
double	失去制导的时间，单位：秒

Remarks

用于判断是否需要触发自毁 超过阈值时可能导致导弹自毁

ThrustAcceleration

获取或设置导弹的推力加速度

Declaration

```
protected Vector3D ThrustAcceleration { get; set; }
```

Property Value

Type	Description
Vector3D	三维加速度向量，单位：米/秒 <sup>2</sup>

Remarks

由发动机产生的推进加速度 影响导弹的速度变化

Methods

Activate()

激活仿真元素，使其参与仿真

Declaration

```
public override void Activate()
```

Overrides

[SimulationElement.Activate\(\)](#)

Remarks

激活操作会：

- 将IsActive设置为true
- 发布实体激活事件
- 允许实体参与仿真计算

## Deactivate()

停用仿真元素，将其从仿真中移除

Declaration

```
public override void Deactivate()
```

Overrides

[SimulationElement.Deactivate\(\)](#)

Remarks

停用操作会：

- 将IsActive设置为false
- 发布实体停用事件
- 停止实体参与仿真计算

## Explode()

导弹爆炸

Declaration

```
public virtual void Explode()
```

Remarks

爆炸过程：

- 停止导弹运动
- 触发爆炸效果
- 发布爆炸事件
- 结束导弹任务

## Fire()

发射导弹

Declaration

```
public virtual void Fire()
```

Remarks

发射过程：

- 激活导弹
- 开始计时和计数

- 启动发动机
- 初始化运动状态

### GetRunningState()

获取导弹的完整运行状态

Declaration

```
public MissileRunningState GetRunningState()
```

Returns

Type	Description
<a href="#">MissileRunningState</a>	包含导弹所有状态信息的结构体

Remarks

返回的状态包括：

- 基本信息（ID、类型）
- 运动状态（位置、速度、朝向）
- 飞行数据（时间、距离）
- 工作状态（制导、发动机）

### GetStatus()

获取导弹的状态信息字符串

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含导弹状态的详细描述

Overrides

[SimulationElement.GetStatus\(\)](#)

Remarks

返回信息包括：

- 导弹ID和类型
- 位置和速度信息
- 飞行时间和距离
- 制导和发动机状态

### SelfDestruct()

导弹自毁

Declaration

```
public void SelfDestruct()
```

Remarks

自毁过程：

- 记录自毁原因
- 停止导弹运动
- 触发自毁效果
- 结束导弹任务

## ShouldSelfDestruct()

检查是否应该自毁

Declaration

```
protected bool ShouldSelfDestruct()
```

Returns

Type	Description
bool	true表示需要自毁，false表示可以继续飞行

Remarks

自毁条件：

- 超出最大飞行时间
- 超出最大飞行距离
- 高度低于安全阈值
- 失去制导时间过长

## Update(double)

更新导弹的状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[SimulationElement.Update\(double\)](#)

Remarks

更新过程：

- 检查导弹是否处于活动状态
- 更新导弹的运动状态
- 更新计时器和计数器

## UpdateGuidanceStatus()

更新导弹的制导状态

Declaration

```
protected virtual void UpdateGuidanceStatus()
```

Remarks

基类中的默认实现为空 具体的导弹类型需要重写此方法 实现各自的制导逻辑

## UpdateMotionState(double)

Declaration

```
protected virtual void UpdateMotionState(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	

## Implements

[IMissile](#)



[Show / Hide Table of Contents](#)

# Class MillimeterWaveDetectionEvent

毫米波探测事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [MillimeterWaveDetectionEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class MillimeterWaveDetectionEvent : SimulationEvent
```

## Properties

### Frequency

毫米波频率(GHz)

#### Declaration

```
public double Frequency { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

### Intensity

毫米波辐射强度

#### Declaration

```
public double Intensity { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## TargetId

目标ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

[Show / Hide Table of Contents](#)

# Class MillimeterWaveWarnerAlarmEvent

毫米波告警器警报事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [MillimeterWaveWarnerAlarmEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class MillimeterWaveWarnerAlarmEvent : SimulationEvent
```

## Properties

### DetectedFrequency

探测到的频率(GHz)

#### Declaration

```
public double DetectedFrequency { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

威胁源仿真库文档

[Back to top](#)

# Namespace ThreatSource.Simulation

## Classes

### EntityActivatedEvent

实体激活事件，表示仿真实体被激活

### EntityDeactivatedEvent

实体停用事件，表示仿真实体被停用

### EntityDestroyedEvent

实体销毁事件，表示仿真实体被销毁

### InfraredDetectionEvent

红外探测事件

### InfraredGuidanceCommandEvent

红外指令制导事件，表示发送制导指令

### InfraredGuidanceMissileLightEvent

红外指令制导导弹点亮红外热源事件

### InfraredGuidanceMissileLightOffEvent

红外指令制导导弹熄灭红外热源事件

### InfraredJammingEvent

红外干扰事件

### InfraredTrackerConfig

红外测角仪配置类

### InfraredWarnerAlarmEvent

红外告警器警报事件

### InfraredWarnerAlarmStopEvent

红外告警器警报停止事件

### InfraredWarnerConfig

红外告警器配置类，用于设置红外探测和告警系统的参数

### **LaserBeamRiderConfig**

激光波束制导仪配置类，用于设置激光波束制导系统的参数

### **LaserBeamStartEvent**

激光波束开始事件，表示激光波束制导开始

### **LaserBeamStopEvent**

激光波束停止事件，表示激光波束制导结束

### **LaserBeamUpdateEvent**

激光波束更新事件，表示激光波束位置更新

### **LaserDesignatorConfig**

激光指示器配置类，用于设置激光半主动导引系统中的激光指示器参数

### **LaserIlluminationStartEvent**

激光照射开始事件，表示激光定位器开始照射目标

### **LaserIlluminationStopEvent**

激光照射停止事件，表示激光定位器停止照射目标

### **LaserIlluminationUpdateEvent**

激光照射更新事件，表示激光照射状态的更新

### **LaserJammerConfig**

激光干扰器配置类

### **LaserJammingEvent**

激光干扰事件，表示对激光制导系统的干扰

### **LaserWarnerAlarmEvent**

激光告警器警报事件，表示检测到激光照射

### **LaserWarnerAlarmStopEvent**

激光告警器警报停止事件，表示激光照射结束

### **LaserWarnerConfig**

激光告警器配置类，用于设置激光探测和告警系统的参数

### **MillimeterWaveDetectionEvent**

毫米波探测事件

### **MillimeterWaveJammerConfig**

毫米波干扰器配置类

## MillimeterWaveJammingEvent

毫米波干扰事件，表示对毫米波雷达的干扰

## MillimeterWaveWarnerAlarmEvent

毫米波告警器警报事件

## MillimeterWaveWarnerAlarmStopEvent

毫米波告警器警报停止事件

## MillimeterWaveWarnerConfig

毫米波告警器配置类，用于设置毫米波探测和告警系统的参数

## MissileFireEvent

导弹发射事件，表示导弹被发射的状态变化

## SimulationElement

仿真元素的抽象基类，所有仿真中的实体都继承自此类

## SimulationEvent

仿真事件的基类，定义所有仿真事件的共同属性

## SimulationManager

仿真管理器的默认实现，提供仿真系统的核心功能

## TankRadiationEvent

坦克辐射事件，表示坦克的多波段辐射特性

## TargetDestroyedEvent

目标被摧毁事件

## TargetHitEvent

目标被击中事件

## UltravioletDetectionEvent

紫外探测事件

## UltravioletWarnerAlarmEvent

紫外告警器警报事件

## UltravioletWarnerAlarmStopEvent

紫外告警器警报停止事件

## UltravioletWarnerConfig

紫外告警器配置类，用于设置紫外探测和告警系统的参数

## Interfaces

**ISimulationAdapter**

第三方仿真环境适配器接口

**ISimulationManager**

仿真管理器接口，提供仿真系统的核心功能



[Show / Hide Table of Contents](#)

# Namespace ThreatSource.Target

## Classes

### Tank

坦克类，实现了目标接口的具体坦克目标

## Interfaces

### ITarget

定义目标对象的基本接口，提供目标的基本特征和属性

# Class InfraredTracker

红外测角仪类，实现了红外制导导弹的跟踪和制导功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [InfraredTracker](#)

## Implements

[IIndicator](#)

## Inherited Members

[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredTracker : SimulationElement, IIndicator
```

## Remarks

该类提供了红外测角仪的核心功能：

- 红外目标跟踪
- 角度测量和计算
- 制导指令生成
- 事件处理和状态管理 通过测量目标的红外辐射实现对导弹的制导控制

## Constructors

### **InfraredTracker(string, string, InfraredTrackerConfig, ISimulationManager)**

初始化红外测角仪的新实例

## Declaration

```
public InfraredTracker(string id, string targetId, InfraredTrackerConfig config, ISimulationManager manager)
```

Parameters

Type	Name	Description
string	id	测角仪ID
string	targetId	目标ID
InfraredTrackerConfig	config	配置参数
ISimulationManager	manager	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性
- 设置目标关联
- 配置工作参数
- 设置初始状态

## Properties

### IsTracking

获取或设置红外测角仪是否正在跟踪目标

Declaration

```
public bool IsTracking { get; }
```

Property Value

Type	Description
bool	

Remarks

true表示正在跟踪目标 false表示未在跟踪状态 用于控制测角仪的工作状态

### MissileId

获取或设置当前跟踪的导弹ID

Declaration

```
public string? MissileId { get; }
```

Property Value

Type	Description
string	

Remarks

记录当前正在跟踪的导弹标识符 用于导弹识别和制导控制

### TargetId

获取或设置目标ID

Declaration

```
public string? TargetId { get; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

记录当前跟踪的目标标识符 用于目标识别和状态更新

## Methods

### Activate()

激活红外测角仪

Declaration

```
public override void Activate()
```

Overrides

[SimulationElement.Activate\(\)](#)

Remarks

激活过程：

- 调用基类激活方法
- 订阅红外事件
- 准备开始工作

### Deactivate()

停用红外测角仪

Declaration

```
public override void Deactivate()
```

Overrides

[SimulationElement.Deactivate\(\)](#)

Remarks

停用过程：

- 调用基类停用方法
- 取消事件订阅
- 清理工作状态

### GetRunningState()

获取红外测角仪运行状态

Declaration

```
public IndicatorRunningState GetRunningState()
```

Returns

Type	Description
------	-------------

Type	Description
<a href="#">IndicatorRunningState</a>	包含测角仪完整状态信息的结构体

Remarks

返回信息包括：

- 目标和导弹关联状态
- 工作类型和状态
- 位置和姿态信息
- 激活状态

GetStatus()

获取红外测角仪状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含测角仪状态的详细描述

Overrides

[SimulationElement.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基本标识信息
- 位置信息
- 跟踪状态
- 目标关联状态
- 工作状态

StopTracking()

停止跟踪目标

Declaration

```
public void StopTracking()
```

Remarks

停止过程：

- 清除导弹关联
- 清除目标关联
- 重置跟踪状态

Update(double)

更新红外测角仪状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
------	------	-------------

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[SimulationElement.Update\(double\)](#)

Remarks

更新过程：

- 检查激活状态
- 更新跟踪状态
- 处理目标跟踪

## Implements

[IIndicator](#)

# Class AltimeterSensorData

测高仪传感器数据类，包含高度测量的具体数据

## Inheritance

↳ [object](#)  
↳ [SensorData](#)  
↳ [AltimeterSensorData](#)

## Inherited Members

[SensorData.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class AltimeterSensorData : SensorData
```

## Remarks

该类封装了毫米波测高仪的测量结果：

- 对地高度信息 用于飞行器高度控制和地形跟随

# Properties

## Altitude

获取或设置测量的高度，单位：米

## Declaration

```
public double Altitude { get; set; }
```

## Property Value

Type	Description
<a href="#">double</a>	

## Remarks

# Class MillimeterWaveAltimeter

毫米波测高雷达类，实现了高度测量和数据采集功能

## Inheritance

↳ [object](#)  
↳ [Sensor](#)  
↳ [MillimeterWaveAltimeter](#)

## Implements

[ISensor](#)

## Inherited Members

[Sensor.IsActive](#)  
[Sensor.Position](#)  
[Sensor.Orientation](#)  
[Sensor.Activate\(\)](#)  
[Sensor.Deactivate\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveAltimeter : Sensor, ISensor
```

## Remarks

该类提供了毫米波测高雷达的核心功能：

- 实时高度测量
- 精度控制
- 干扰监测
- 数据采集 用于末敏子弹的高度测量和地形跟踪

## Constructors

### **MillimeterWaveAltimeter(TerminalSensitiveSubmunition, double, double)**

初始化毫米波测高雷达的新实例

## Declaration



```
public MillimeterWaveAltimeter(TerminalSensitiveSubmunition submunition, double maxAltitude, double accuracy)
```

Parameters

Type	Name	Description
<a href="#">TerminalSensitiveSubmunition</a>	<i>submunition</i>	末敏子弹实例
<a href="#">double</a>	<i>maxAltitude</i>	最大测量高度，单位：米
<a href="#">double</a>	<i>accuracy</i>	测量精度，单位：米

Remarks

构造过程：

- 设置量程参数
- 初始化高度记录
- 继承基类位置和姿态

## Properties

### Accuracy

获取或设置测量精度，单位：米

Declaration

```
public double Accuracy { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

定义了高度测量的误差范围 实际测量值在真实高度±精度范围内

### MaxAltitude

获取或设置最大测量高度，单位：米

Declaration

```
public double MaxAltitude { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

定义了测高雷达的量程上限 超出此高度的测量可能不准确

## Methods

### GetSensorData()

获取毫米波测高雷达的最新数据

Declaration

```
public override SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含高度测量结果的数据对象

Overrides

[Sensor.GetSensorData\(\)](#)

Remarks

- 返回数据：
- 当前高度值
  - 测量时间戳
  - 数据可靠性

OnMillimeterWaveJamming(object, EventArgs)

处理毫米波干扰事件

Declaration

```
public void OnMillimeterWaveJamming(object sender, EventArgs e)
```

Parameters

Type	Name	Description
<a href="#">object</a>	<i>sender</i>	事件源对象
<a href="#">EventArgs</a>	<i>e</i>	事件参数

Remarks

- 当检测到毫米波干扰时：
- 输出警告信息
  - 可能影响测量精度
  - 需要采取抗干扰措施

Update(double)

更新毫米波测高雷达的工作状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Overrides

[Sensor.Update\(double\)](#)

Remarks

- 更新过程：
- 获取当前位置高度
  - 添加测量精度误差
  - 更新高度记录

# Implements

ISensor

威胁源仿真库文档

[Back to top](#)

# Class TargetHitEvent

目标被击中事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [TargetHitEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class TargetHitEvent : SimulationEvent
```

## Properties

### MissileId

导弹ID

#### Declaration

```
public string? MissileId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

威胁源仿真库文档

[Back to top](#)

# Class LaserSemiActiveGuidedMissile

激光半主动制导导弹类，实现了激光半主动寻的制导功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [LaserSemiActiveGuidedMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Explode\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class LaserSemiActiveGuidedMissile : BaseMissile, IMissile
```

Remarks

- 该类提供了激光半主动制导导弹的核心功能：
- 激光照射目标跟踪
  - 半主动制导控制
  - 飞行阶段管理
  - 状态监控和事件处理 通过跟踪目标反射的激光能量实现对导弹的制导控制

Constructors

LaserSemiActiveGuidedMissile(MissileProperties, ISimulationManager)

初始化激光半主动制导导弹的新实例

Declaration

```
public LaserSemiActiveGuidedMissile(MissileProperties config, ISimulationManager manager)
```

Parameters

Type	Name	Description
MissileProperties	config	导弹配置参数
ISimulationManager	manager	仿真管理器实例

Remarks

- 构造过程：
- 初始化基本属性
  - 创建制导系统
  - 配置制导参数
  - 设置初始飞行阶段

Properties

LaserDesignatorId

获取或设置激光指示器ID

Declaration

```
public string LaserDesignatorId { get; set; }
```

Property Value

Type	Description
string	激光指示器的唯一标识符

Remarks

用于识别和关联激光照射源 影响导弹的制导精度

Methods

Activate()

激活导弹

Declaration

```
public override void Activate()
```

Overrides

[BaseMissile.Activate\(\)](#)

Remarks

激活过程：

- 调用基类激活方法
- 订阅激光照射事件
- 准备目标跟踪

Deactivate()

停用导弹

Declaration

```
public override void Deactivate()
```

Overrides

[BaseMissile.Deactivate\(\)](#)

Remarks

停用过程：

- 调用基类停用方法
- 取消订阅激光照射事件
- 清理制导状态

GetStatus()

获取导弹状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基本状态信息
- 制导系统状态
- 目标跟踪状态

Update(double)

更新导弹状态

Declaration



```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

- 根据当前阶段更新状态
- 处理阶段转换
- 调用基类更新

## Implements

[IMissile](#)

[Show / Hide Table of Contents](#)

# Class UltravioletWarnerAlarmStopEvent

紫外告警器警报停止事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ UltravioletWarnerAlarmStopEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class UltravioletWarnerAlarmStopEvent : SimulationEvent
```

## Properties

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Class SimulationEvent

仿真事件的基类，定义所有仿真事件的共同属性

## Inheritance

↳ [object](#)

↳ [SimulationEvent](#)

- ↳ [EntityActivatedEvent](#)
- ↳ [EntityDeactivatedEvent](#)
- ↳ [EntityDestroyedEvent](#)
- ↳ [InfraredDetectionEvent](#)
- ↳ [InfraredGuidanceCommandEvent](#)
- ↳ [InfraredGuidanceMissileLightEvent](#)
- ↳ [InfraredGuidanceMissileLightOffEvent](#)
- ↳ [InfraredJammingEvent](#)
- ↳ [InfraredWarnerAlarmEvent](#)
- ↳ [InfraredWarnerAlarmStopEvent](#)
- ↳ [LaserBeamStartEvent](#)
- ↳ [LaserBeamStopEvent](#)
- ↳ [LaserBeamUpdateEvent](#)
- ↳ [LaserIlluminationStartEvent](#)
- ↳ [LaserIlluminationStopEvent](#)
- ↳ [LaserIlluminationUpdateEvent](#)
- ↳ [LaserJammingEvent](#)
- ↳ [LaserWarnerAlarmEvent](#)
- ↳ [LaserWarnerAlarmStopEvent](#)
- ↳ [MillimeterWaveDetectionEvent](#)
- ↳ [MillimeterWaveJammingEvent](#)
- ↳ [MillimeterWaveWarnerAlarmEvent](#)
- ↳ [MillimeterWaveWarnerAlarmStopEvent](#)
- ↳ [MissileFireEvent](#)
- ↳ [TankRadiationEvent](#)
- ↳ [TargetDestroyedEvent](#)
- ↳ [TargetHitEvent](#)
- ↳ [UltravioletDetectionEvent](#)
- ↳ [UltravioletWarnerAlarmEvent](#)
- ↳ [UltravioletWarnerAlarmStopEvent](#)

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

Syntax

```
public class SimulationEvent
```

**Remarks**

所有仿真事件都继承自此类，提供：

- 事件发送者标识
- 事件发生时间戳 用于在仿真系统中追踪和处理各类事件

# Properties

## SenderId

获取或设置事件发送者的ID

Declaration

```
public string? SenderId { get; set; }
```

Property Value

Type	Description
string	

**Remarks**

用于标识事件的来源实体 可以为null，表示系统事件

## Timestamp

获取或设置事件发生的时间戳

Declaration

```
public double Timestamp { get; set; }
```

Property Value

Type	Description
double	

**Remarks**

使用UTC时间的Ticks值 用于事件的时序管理和同步

# Interface IMissile

导弹接口，定义了导弹的基本行为和状态

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface IMissile
```

## Remarks

该接口定义了导弹的核心功能：

- 制导状态管理
- 发射控制
- 爆炸和自毁功能
- 运行状态查询 所有具体的导弹类型都应实现此接口

## Properties

### IsGuidance

获取导弹是否处于制导状态

Declaration

```
bool IsGuidance { get; }
```

Property Value

Type	Description
<a href="#">bool</a>	

Remarks

true表示导弹当前处于制导阶段 false表示导弹处于非制导飞行状态 用于控制导弹的制导系统工作状态

## Methods

### Explode()

引爆导弹，触发爆炸效果

Declaration

```
void Explode()
```

Remarks

爆炸过程包括：

- 计算爆炸范围和伤害
- 对周围目标造成伤害
- 触发爆炸事件
- 导弹自身销毁

Fire()

发射导弹，使导弹进入工作状态

Declaration

```
void Fire()
```

Remarks

发射过程包括：

- 点火启动发动机
- 初始化制导系统
- 开始飞行状态计算
- 触发导弹发射事件

GetRunningState()

获取导弹的当前运行状态信息

Declaration

```
MissileRunningState GetRunningState()
```

Returns

Type	Description
<a href="#">MissileRunningState</a>	包含导弹完整状态信息的结构体

Remarks

返回的状态信息包括：

- 基本信息（ID、类型）
- 运动状态（位置、速度、朝向）
- 飞行数据（时间、距离）
- 工作状态（制导、发动机）

SelfDestruct()

执行导弹自毁程序

Declaration

```
void SelfDestruct()
```

Remarks

自毁触发条件：

- 超出最大飞行时间
- 超出最大飞行距离
- 失去制导时间过长
- 接收到自毁指令 自毁会立即停止导弹的工作并销毁

[Show / Hide Table of Contents](#)

# Class InfraredWarnerAlarmEvent

红外告警器警报事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ InfraredWarnerAlarmEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredWarnerAlarmEvent : SimulationEvent
```

## Properties

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Class Tank

坦克类，实现了目标接口的具体坦克目标

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ Tank

## Implements

[ITarget](#)

## Inherited Members

[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.GetStatus\(\)](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[SimulationElement.Activate\(\)](#)  
[SimulationElement.Deactivate\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Target](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class Tank : SimulationElement, ITarget
```

## Remarks

该类提供了坦克目标的完整实现：

- 继承自SimulationElement，具备基本的仿真功能
- 实现ITarget接口，提供目标特征
- 包含生命值系统，可以响应伤害
- 支持位置更新和状态同步

## Constructors

**Tank(string, Vector3D, double, ISimulationManager)**



初始化坦克类的新实例

Declaration

```
public Tank(string id, Vector3D position, double speed, ISimulationManager simulationManager)
```

Parameters

Type	Name	Description
string	id	坦克的唯一标识符
Vector3D	position	初始位置坐标
double	speed	初始速度，单位：米/秒
ISimulationManager	simulationManager	仿真管理器实例

Remarks

构造函数设置坦克的初始状态：

- 使用默认朝向
- 设置初始位置和速度
- 生命值初始化为100

Properties

Health

获取或设置坦克的当前生命值

Declaration

```
public double Health { get; set; }
```

Property Value

Type	Description
double	

Remarks

范围：0-100 初始值为100 当生命值降至0或以下时，坦克被销毁

InfraredRadiationIntensity

获取坦克的红外辐射强度

Declaration

```
public double InfraredRadiationIntensity { get; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特/平方米 固定值100.0，表示坦克发动机和装甲的热辐射特性

RadarCrossSection

获取坦克的雷达散射截面积

Declaration

```
public double RadarCrossSection { get; }
```

Property Value

Type	Description
double	

Remarks

单位：平方米 固定值10.0，表示标准坦克的雷达反射特性

Type

获取坦克的类型标识

Declaration

```
public string Type { get; }
```

Property Value

Type	Description
string	

Remarks

固定返回"Tank"，用于标识这是一个坦克目标

Methods

TakeDamage(double)

对坦克造成伤害

Declaration

```
public void TakeDamage(double damage)
```

Parameters

Type	Name	Description
double	damage	伤害值

Remarks

伤害处理过程：

- 从当前生命值中扣除伤害值
- 如果生命值降至0或以下，触发目标销毁事件
- 销毁事件会通知仿真系统移除该目标

Update(double)

更新坦克的状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[SimulationElement.Update\(double\)](#)

Remarks

更新过程：

- 根据当前速度更新位置
- 位置更新使用简单的线性运动模型

## Implements

[ITarget](#)

# Class LaserIlluminationStopEvent

激光照射停止事件，表示激光定位器停止照射目标

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserIlluminationStopEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserIlluminationStopEvent : SimulationEvent
```

## Remarks

用于通知系统激光照射已结束 触发时机：激光定位器停止照射或照射中断时

## Properties

### LaserDesignatorId

获取或设置激光定位器的ID

#### Declaration

```
public string? LaserDesignatorId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

### TargetId

获取或设置被照射目标的ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
string	

# Namespace ThreatSource.Sensor

## Classes

### **AltimeterSensorData**

测高仪传感器数据类，包含高度测量的具体数据

### **InfraredDetector**

红外探测器类，实现了红外辐射的探测和目标识别功能

### **InfraredSensorData**

红外传感器数据类，包含红外探测的具体数据

### **LaserRangefinder**

激光测距仪类，实现了目标距离测量和数据采集功能

### **MillimeterWaveAltimeter**

毫米波测高雷达类，实现了高度测量和数据采集功能

### **MillimeterWaveRadiometer**

毫米波辐射计类，实现了目标辐射温度测量和目标识别功能

### **RadiometerSensorData**

辐射计传感器数据类，包含毫米波辐射探测的具体数据

### **RangefinderSensorData**

测距仪传感器数据类，包含距离测量的具体数据

### **Sensor**

传感器的抽象基类，实现了ISensor接口的基本功能

### **SensorData**

传感器数据的抽象基类，定义了所有传感器数据的通用属性

## Interfaces

### **ISensor**

[Show / Hide Table of Contents](#)

# Interface IIndicator

指示器接口，定义了所有指示器的通用功能

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface IIndicator
```

## Remarks

该接口提供了指示器的基本功能：

- 目标和导弹的关联
- 运行状态的获取
- 位置和姿态的管理 是激光指示器、红外跟踪器等具体指示器的抽象基础

## Properties

### MissileId

获取或设置导弹ID

Declaration

```
string? MissileId { get; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

用于标识和控制关联的导弹 可以为null表示当前没有关联导弹

### TargetId

获取或设置目标ID

Declaration

```
string? TargetId { get; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

用于标识和跟踪目标 可以为null表示当前没有关联目标

Methods

GetRunningState()

获取指示器的当前运行状态

Declaration

```
IndicatorRunningState GetRunningState()
```

Returns

Type	Description
<a href="#">IndicatorRunningState</a>	包含指示器完整状态信息的结构体

Remarks

返回的状态信息包括：

- 目标和导弹的关联状态
- 指示器的工作状态
- 位置、速度和姿态信息
- 激活状态



# Namespace ThreatSource.Indicator

## Classes

### **InfraredTracker**

红外测角仪类，实现了红外制导导弹的跟踪和制导功能

### **LaserBeamRider**

激光波束制导器类，用于生成激光波束制导场

### **LaserDesignator**

激光指示器类，实现了对目标的激光照射和制导功能

## Structs

### **IndicatorRunningState**

指示器运行状态结构体，包含了指示器的完整状态信息

## Interfaces

### **IIndicator**

指示器接口，定义了所有指示器的通用功能

## Enums

### **IndicatorState**

指示器状态枚举，定义了指示器的工作状态

### **IndicatorType**

指示器类型枚举，定义了支持的指示器类型

[Show / Hide Table of Contents](#)

# Class InfraredImagingTerminalGuidedMissile

红外成像末制导导弹类，继承自基础导弹类

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [InfraredImagingTerminalGuidedMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Activate\(\)](#)  
[BaseMissile.Deactivate\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class InfraredImagingTerminalGuidedMissile : BaseMissile, IMissile
```

Remarks

该类提供了红外成像末制导导弹的完整实现：

- 继承自BaseMissile，具备基本的导弹功能
- 实现了红外成像制导系统
- 支持多阶段飞行控制
- 具备红外目标识别能力
- 提供精确的末制导打击能力

工作流程：

1. 导弹发射并进入发射阶段
2. 切换到巡航阶段，进行中程飞行
3. 进入末制导阶段，启动红外成像制导
4. 接近目标后引爆或达到自毁条件后自毁

Constructors

InfraredImagingTerminalGuidedMissile(MissileProperties, ISimulationManager)

初始化红外成像末制导导弹的新实例

Declaration

```
public InfraredImagingTerminalGuidedMissile(MissileProperties config, ISimulationManager manager)
```

Parameters

Type	Name	Description
<a href="#">MissileProperties</a>	<i>config</i>	导弹的配置参数
<a href="#">ISimulationManager</a>	<i>manager</i>	仿真管理器实例

Remarks

构造过程：

1. 调用基类构造函数
2. 创建制导系统实例
3. 设置初始飞行阶段

Methods

Explode()

执行导弹爆炸

Declaration

```
public override void Explode()
```

Overrides

[BaseMissile.Explode\(\)](#)

Remarks

爆炸过程：

1. 调用基类的爆炸方法
2. 切换到爆炸阶段

### GetStatus()

获取导弹的状态信息字符串

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基类状态信息
- 当前飞行阶段
- 制导系统状态
- 目标跟踪信息

### Update(double)

更新导弹的状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

1. 调用基类的更新方法
2. 根据当前阶段选择更新方法
3. 执行相应阶段的更新逻辑
4. 检查是否需要自毁

### Implements

[IMissile](#)

# Class LaserBeamRiderGuidanceSystem

激光驾束制导系统类，实现了基于激光束跟踪的制导功能

## Inheritance

↳ [object](#)  
↳ [BasicGuidanceSystem](#)  
↳ [LaserBeamRiderGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[BasicGuidanceSystem.HasGuidance](#)  
[BasicGuidanceSystem.MaxAcceleration](#)  
[BasicGuidanceSystem.ProportionalNavigationCoefficient](#)  
[BasicGuidanceSystem.Position](#)  
[BasicGuidanceSystem.Velocity](#)  
[BasicGuidanceSystem.GuidanceAcceleration](#)  
[BasicGuidanceSystem.GetGuidanceAcceleration\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserBeamRiderGuidanceSystem : BasicGuidanceSystem, IGuidanceSystem
```

## Remarks

该类提供了激光驾束制导系统的核心功能：

- 激光束生成和控制
- 偏差检测和计算
- PID控制器
- 非线性增益控制 用于实现高精度的制导控制

## Constructors

### LaserBeamRiderGuidanceSystem(double, double)

初始化激光驾束制导系统的新实例

## Declaration

```
public LaserBeamRiderGuidanceSystem(double maxAcceleration, double guidanceCoefficient)
```

Parameters

Type	Name	Description
double	<i>maxAcceleration</i>	最大加速度，单位：米/平方秒
double	<i>guidanceCoefficient</i>	制导系数

Remarks

构造过程：

- 初始化基类参数
- 初始化激光参数
- 初始化控制参数

## Properties

### LaserPower

获取或设置激光功率，单位：瓦特

Declaration

```
public double LaserPower { get; set; }
```

Property Value

Type	Description
double	

Remarks

记录激光源的发射功率 影响系统的探测距离和可靠性

### LastGuidanceAcceleration

获取上一次的制导加速度

Declaration

```
public Vector3D LastGuidanceAcceleration { get; }
```

Property Value

Type	Description
Vector3D	

Remarks

记录历史制导指令 用于实现低通滤波

## Methods

### CalculateGuidanceAcceleration(double)

计算制导加速度

Declaration

```
protected override void CalculateGuidanceAcceleration(double deltaTime)
```

Parameters

Type	Name	Description
double	deltaTime	时间步长，单位：秒

Overrides

[BasicGuidanceSystem.CalculateGuidanceAcceleration\(double\)](#)

Remarks

计算过程：

- 计算偏差向量
- 执行PID控制
- 应用非线性增益
- 计算横向加速度
- 计算前向加速度
- 合成最终制导指令
- 应用低通滤波

## DeactivateLaserBeam()

关闭激光照射系统

Declaration

```
public void DeactivateLaserBeam()
```

Remarks

关闭过程：

- 清除位置信息
- 清除方向信息
- 清除功率参数
- 停止制导

## GetStatus()

获取制导系统的详细状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含完整状态参数的字符串

Overrides

[BasicGuidanceSystem.GetStatus\(\)](#)

Remarks

返回信息：

- 基本状态信息
- 制导参数
- 控制状态 用于系统监控和调试

## Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

Declaration

```
public override void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

Parameters

Type	Name	Description
double	deltaTime	时间步长，单位：秒
Vector3D	missilePosition	导弹位置，单位：米
Vector3D	missileVelocity	导弹速度，单位：米/秒

Overrides

[BasicGuidanceSystem.Update\(double, Vector3D, Vector3D\)](#)

Remarks

更新过程：

- 检查激光照射状态
- 更新制导状态
- 计算制导指令
- 更新输出参数

## UpdateLaserBeamRider(Vector3D, Vector3D, double)

更新激光驾束仪参数

Declaration

```
public void UpdateLaserBeamRider(Vector3D sourcePosition, Vector3D direction, double laserPower)
```

Parameters

Type	Name	Description
Vector3D	sourcePosition	激光源位置，单位：米
Vector3D	direction	激光方向向量
double	laserPower	激光功率，单位：瓦特

Remarks

更新过程：

- 激活激光照射
- 更新位置信息
- 更新方向信息
- 更新功率参数

## Implements

[IGuidanceSystem](#)



# Class InfraredImagingGuidanceSystem

红外成像引导系统类，实现了基于红外图像的目标探测和跟踪功能

## Inheritance

↳ [object](#)  
↳ [BasicGuidanceSystem](#)  
↳ [InfraredImagingGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[BasicGuidanceSystem.HasGuidance](#)  
[BasicGuidanceSystem.MaxAcceleration](#)  
[BasicGuidanceSystem.ProportionalNavigationCoefficient](#)  
[BasicGuidanceSystem.Position](#)  
[BasicGuidanceSystem.Velocity](#)  
[BasicGuidanceSystem.GuidanceAcceleration](#)  
[BasicGuidanceSystem.GetGuidanceAcceleration\(\)](#)  
[BasicGuidanceSystem.CalculateGuidanceAcceleration\(double\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredImagingGuidanceSystem : BasicGuidanceSystem, IGuidanceSystem
```

## Remarks

该类提供了红外成像制导系统的核心功能：

- 目标自动探测
- 图像识别处理
- 信噪比计算
- 比例导引控制 用于实现自主寻的制导

## Constructors

### **InfraredImagingGuidanceSystem(double, double, ISimulationManager)**

初始化红外成像引导系统的新实例

## Declaration

```
public InfraredImagingGuidanceSystem(double maxAcceleration, double proportionalNavigationCoefficient, ISimulationManager simulationManager)
```

Parameters

Type	Name	Description
double	maxAcceleration	最大加速度，单位：米/平方秒
double	proportionalNavigationCoefficient	比例导引系数
ISimulationManager	simulationManager	仿真管理器实例

Remarks

构造过程：

- 初始化基类参数
- 设置仿真管理器
- 初始化目标位置

## Properties

### SimulationManager

获取或设置仿真管理器实例

Declaration

```
public ISimulationManager SimulationManager { get; set; }
```

Property Value

Type	Description
ISimulationManager	

Remarks

用于获取场景中的目标信息 实现目标探测功能

## Methods

### GetStatus()

获取制导系统的详细状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含完整状态参数的字符串

Overrides

[BasicGuidanceSystem.GetStatus\(\)](#)

Remarks

返回信息：

- 基本状态信息
- 制导加速度

- 目标位置 用于系统监控和调试

## Update(double, Vector3D, Vector3D)

更新引导系统的状态和计算结果

### Declaration

```
public override void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

### Parameters

Type	Name	Description
double	deltaTime	自上次更新以来的时间间隔，单位：秒
Vector3D	missilePosition	导弹当前位置，单位：米
Vector3D	missileVelocity	导弹当前速度，单位：米/秒

### Overrides

[BasicGuidanceSystem.Update\(double, Vector3D, Vector3D\)](#)

### Remarks

更新过程：

- 探测目标位置
- 计算目标速度
- 生成制导指令
- 限制最大加速度

## Implements

[IGuidanceSystem](#)

[Show / Hide Table of Contents](#)

# Class UltravioletWarnerConfig

紫外告警器配置类，用于设置紫外探测和告警系统的参数

## Inheritance

↳ [object](#)

↳ UltravioletWarnerConfig

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class UltravioletWarnerConfig
```

## Remarks

该类定义了紫外告警器的关键参数：

- 设备标识
- 灵敏度阈值
- 警报持续时间
- 探测波长范围 这些参数决定了告警器对紫外辐射的探测能力

## Constructors

### UltravioletWarnerConfig()

初始化紫外告警器配置的新实例

## Declaration

```
public UltravioletWarnerConfig()
```

## Remarks

设置默认值：

- ID为空字符串
- 灵敏度阈值为0
- 警报持续时间为5秒
- 波长范围为0-0纳米

# Properties

## AlarmDuration

获取或设置警报持续时间

Declaration

```
public double AlarmDuration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 定义了每次告警的持续时间

## Id

获取或设置紫外告警器的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的告警器

## SensitivityThreshold

获取或设置告警灵敏度阈值

Declaration

```
public double SensitivityThreshold { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特/平方米 当接收到的紫外辐射强度超过此阈值时触发告警

## WavelengthMax

获取或设置最大探测波长

Declaration

```
public double WavelengthMax { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最长紫外波长

WavelengthMin

获取或设置最小探测波长

Declaration

```
public double WavelengthMin { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最短紫外波长

[Show / Hide Table of Contents](#)

# Class BasicGuidanceSystem

基础制导系统类，实现了制导系统的基本功能框架

## Inheritance

↳ [object](#)

↳ [BasicGuidanceSystem](#)

↳ [InfraredCommandGuidanceSystem](#)

↳ [InfraredImagingGuidanceSystem](#)

↳ [LaserBeamRiderGuidanceSystem](#)

↳ [LaserSemiActiveGuidanceSystem](#)

↳ [MillimeterWaveGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class BasicGuidanceSystem : IGuidanceSystem
```

## Remarks

该类提供了制导系统的通用实现：

- 制导状态管理
- 运动参数记录
- 加速度限制
- 比例导引系数 是其他具体制导系统的基类

## Constructors

### BasicGuidanceSystem(double, double)

初始化基础制导系统的新实例

#### Declaration

```
public BasicGuidanceSystem(double maxAcceleration, double proportionalNavigationCoefficient)
```

Parameters

Type	Name	Description
double	maxAcceleration	最大加速度，单位：米/平方秒
double	proportionalNavigationCoefficient	比例导引系数

Remarks

构造过程：

- 初始化制导状态
- 设置运动参数
- 配置制导参数

Properties

GuidanceAcceleration

获取或设置制导加速度，单位：米/平方秒

Declaration

```
protected Vector3D GuidanceAcceleration { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

存储计算得到的制导指令 用于控制导弹的运动轨迹

HasGuidance

获取或设置是否有有效的制导信息

Declaration

```
public bool HasGuidance { get; protected set; }
```

Property Value

Type	Description
bool	

Remarks

true表示当前有可用的制导信息 false表示无法获得有效制导 用于判断制导系统的工作状态

MaxAcceleration

获取或设置最大加速度，单位：米/平方秒

Declaration

```
public double MaxAcceleration { get; set; }
```

Property Value

Type	Description
double	



Remarks  
定义了制导加速度的上限 用于限制导弹的机动能力 考虑结构和动力限制

Position

获取或设置导弹位置，单位：米

Declaration

```
protected Vector3D Position { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks  
记录导弹的当前位置 用于制导计算和状态更新

ProportionalNavigationCoefficient

获取或设置比例导引系数

Declaration

```
public double ProportionalNavigationCoefficient { get; set; }
```

Property Value

Type	Description
double	

Remarks  
定义了制导指令的增益 影响制导系统的响应特性 通常取值3~5

Velocity

获取或设置导弹速度，单位：米/秒

Declaration

```
protected Vector3D Velocity { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks  
记录导弹的当前速度 用于制导计算和状态更新

Methods

CalculateGuidanceAcceleration(double)

计算制导加速度（需要在子类中实现）

Declaration

```
protected virtual void CalculateGuidanceAcceleration(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间间隔，单位：秒

Remarks

计算要求：

- 实现特定的制导律
- 考虑最大加速度限制
- 更新制导加速度 基类中不实现具体计算

## GetGuidanceAcceleration()

获取制导加速度指令

Declaration

```
public Vector3D GetGuidanceAcceleration()
```

Returns

Type	Description
Vector3D	三维制导加速度向量，单位：米/平方秒

Remarks

返回数据：

- X分量：横向制导加速度
- Y分量：垂直制导加速度
- Z分量：纵向制导加速度 用于导弹的轨迹控制

## GetStatus()

获取制导系统的状态信息

Declaration

```
public virtual string GetStatus()
```

Returns

Type	Description
string	包含关键状态参数的字符串

Remarks

返回信息：

- 制导状态
- 位置信息
- 速度信息
- 加速度信息 用于状态监控和调试

## Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

Declaration

```
public virtual void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

#### Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒
<a href="#">Vector3D</a>	<i>missilePosition</i>	导弹当前位置，单位：米
<a href="#">Vector3D</a>	<i>missileVelocity</i>	导弹当前速度，单位：米/秒

#### Remarks

更新过程：

- 更新位置信息
- 更新速度信息
- 准备制导计算

## Implements

[IGuidanceSystem](#)

# Class MissileFireEvent

导弹发射事件，表示导弹被发射的状态变化

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [MissileFireEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class MissileFireEvent : SimulationEvent
```

## Remarks

在导弹发射时触发，包含目标信息 用于通知其他实体和系统导弹已发射

# Properties

## TargetId

获取或设置目标实体的ID

## Declaration

```
public string? TargetId { get; set; }
```

## Property Value

Type	Description
<a href="#">string</a>	

## Remarks

标识导弹的预定目标 可以为null 表示无预定目标

[Show / Hide Table of Contents](#)

# Interface ISensor

定义传感器的基本接口，规范了所有传感器的通用功能

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface ISensor
```

## Remarks

该接口提供了传感器的基本功能规范：

- 激活和停用控制
- 状态更新机制
- 数据采集接口 是所有具体传感器实现的基础

## Properties

### IsActive

获取或设置传感器是否处于激活状态

Declaration

```
bool IsActive { get; set; }
```

Property Value

Type	Description
<a href="#">bool</a>	

Remarks

true表示传感器正在工作 false表示传感器处于待机状态 用于控制传感器的工作状态

## Methods

### Activate()

激活传感器，使其开始工作

Declaration

```
void Activate()
```

Remarks

激活过程：

- 初始化工作参数
- 启动数据采集
- 开始状态监控

Deactivate()

停用传感器，使其停止工作

Declaration

```
void Deactivate()
```

Remarks

停用过程：

- 停止数据采集
- 清理工作状态
- 释放相关资源

GetSensorData()

获取传感器采集的最新数据

Declaration

```
SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含传感器测量结果的数据对象

Remarks

返回数据：

- 测量的物理量
- 时间戳信息
- 状态标志

Update(double)

更新传感器的工作状态

Declaration

```
void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Remarks

更新过程：

- 采集最新数据
- 更新内部状态
- 处理异常情况

[Show / Hide Table of Contents](#)

# Class LaserWarnerAlarmStopEvent

激光告警器警报停止事件，表示激光照射结束

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserWarnerAlarmStopEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserWarnerAlarmStopEvent : SimulationEvent
```

## Remarks

用于模拟激光告警器停止报警 触发时机：激光照射结束或低于警戒阈值时

## Properties

### TargetId

获取或设置目标的ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识激光照射已结束的实体

# Class LaserBeamRiderMissile

激光波束制导导弹类，实现了激光波束跟踪制导的导弹功能

## Inheritance

↳ [object](#)  
↳ [SimulationElement](#)  
↳ [BaseMissile](#)  
↳ [LaserBeamRiderMissile](#)

## Implements

[IMissile](#)

## Inherited Members

[BaseMissile.FlightTime](#)  
[BaseMissile.FlightDistance](#)  
[BaseMissile.EngineBurnTime](#)  
[BaseMissile.IsGuidance](#)  
[BaseMissile.LostGuidanceTime](#)  
[BaseMissile.LastKnownVelocity](#)  
[BaseMissile.GuidanceAcceleration](#)  
[BaseMissile.ThrustAcceleration](#)  
[BaseMissile.MissileProperties](#)  
[BaseMissile.UpdateMotionState\(double\)](#)  
[BaseMissile.Fire\(\)](#)  
[BaseMissile.ShouldSelfDestruct\(\)](#)  
[BaseMissile.Explode\(\)](#)  
[BaseMissile.SelfDestruct\(\)](#)  
[BaseMissile.UpdateGuidanceStatus\(\)](#)  
[BaseMissile.GetRunningState\(\)](#)  
[SimulationElement.Id](#)  
[SimulationElement.Position](#)  
[SimulationElement.Speed](#)  
[SimulationElement.Velocity](#)  
[SimulationElement.Orientation](#)  
[SimulationElement.IsActive](#)  
[SimulationElement.SimulationManager](#)  
[SimulationElement.PublishEvent\(SimulationEvent\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax



```
public class LaserBeamRiderMissile : BaseMissile, IMissile
```

Remarks

该类提供了激光波束制导导弹的核心功能：

- 激光波束跟踪
- 波束制导控制
- 飞行阶段管理
- 状态监控和事件处理 通过跟踪激光波束的中心线实现对导弹的制导控制

Constructors

LaserBeamRiderMissile(MissileProperties, ISimulationManager)

初始化激光波束制导导弹的新实例

Declaration

```
public LaserBeamRiderMissile(MissileProperties config, ISimulationManager manager)
```

Parameters

Type	Name	Description
<a href="#">MissileProperties</a>	<i>config</i>	导弹配置参数
<a href="#">ISimulationManager</a>	<i>manager</i>	仿真管理器实例

Remarks

构造过程：

- 初始化基本属性
- 创建波束制导系统
- 配置制导参数
- 设置初始飞行阶段

Methods

Activate()

激活导弹

Declaration

```
public override void Activate()
```

Overrides

[BaseMissile.Activate\(\)](#)

Remarks

激活过程：

- 调用基类激活方法
- 订阅激光波束事件
- 准备波束跟踪

Deactivate()

停用导弹

Declaration

```
public override void Deactivate()
```

Overrides

[BaseMissile.Deactivate\(\)](#)

Remarks

停用过程：

- 调用基类停用方法
- 取消订阅波束事件
- 清理制导状态

## GetStatus()

获取导弹状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含导弹状态的详细描述

Overrides

[BaseMissile.GetStatus\(\)](#)

Remarks

返回信息包括：

- 基本状态信息
- 波束跟踪状态
- 制导系统状态

## Update(double)

更新导弹状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BaseMissile.Update\(double\)](#)

Remarks

更新过程：

- 根据当前阶段更新状态
- 处理阶段转换
- 调用基类更新

## Implements

[IMissile](#)

# Struct IndicatorRunningState

指示器运行状态结构体，包含了指示器的完整状态信息

## Inherited Members

[ValueType.Equals\(object\)](#)  
[ValueType.GetHashCode\(\)](#)  
[ValueType.ToString\(\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetType\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public struct IndicatorRunningState
```

## Remarks

该结构体封装了指示器的所有关键状态：

- 关联信息（目标和导弹）
- 工作状态（类型和状态）
- 运动状态（位置、速度、姿态）
- 功能状态（是否激活） 用于状态监控和控制决策

## Properties

### IsActive

获取或设置指示器是否激活

#### Declaration

```
public bool IsActive { readonly get; set; }
```

#### Property Value

Type	Description
<a href="#">bool</a>	

#### Remarks

表示指示器是否处于工作状态 true表示正在工作，false表示待机

### MissileId

获取或设置导弹ID

Declaration

```
public string MissileId { readonly get; set; }
```

Property Value

Type	Description
string	

Remarks

标识当前控制的导弹 用于导弹制导和状态同步

Orientation

获取或设置指示器朝向

Declaration

```
public Orientation Orientation { readonly get; set; }
```

Property Value

Type	Description
Orientation	

Remarks

指示器在三维空间中的姿态 影响指示器的工作效果和精度

Position

获取或设置指示器位置

Declaration

```
public Vector3D Position { readonly get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

指示器在三维空间中的位置向量 用于空间定位和运动控制

State

获取或设置指示器状态

Declaration

```
public IndicatorState State { readonly get; set; }
```

Property Value

Type	Description
IndicatorState	

Remarks

表示指示器的当前工作状态 用于控制指示器的工作模式

## TargetId

获取或设置目标ID

Declaration

```
public string TargetId { readonly get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

标识当前跟踪的目标 用于目标关联和状态更新

## Type

获取或设置指示器类型

Declaration

```
public IndicatorType Type { readonly get; set; }
```

Property Value

Type	Description
<a href="#">IndicatorType</a>	

Remarks

表示指示器的具体类型 影响指示器的工作模式和性能参数

## Velocity

获取或设置指示器速度

Declaration

```
public Vector3D Velocity { readonly get; set; }
```

Property Value

Type	Description
<a href="#">Vector3D</a>	

Remarks

指示器在三维空间中的速度向量 用于运动状态更新和预测

# Class EntityDeactivatedEvent

实体停用事件，表示仿真实体被停用

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ EntityDeactivatedEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class EntityDeactivatedEvent : SimulationEvent
```

## Remarks

用于通知系统某个实体已停止活动 触发时机：实体被停用或暂时移除时

## Properties

### DeactivatedEntityId

获取或设置被停用实体的ID

#### Declaration

```
public string? DeactivatedEntityId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识已被停用的实体

# Class LaserBeamRiderConfig

激光波束制导仪配置类，用于设置激光波束制导系统的参数

## Inheritance

↳ [object](#)

↳ LaserBeamRiderConfig

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserBeamRiderConfig
```

## Remarks

该类定义了波束制导系统的关键参数：

- 设备标识
- 发射位置
- 激光功率
- 控制场尺寸
- 最大制导距离 这些参数决定了波束制导系统的性能和工作范围

## Constructors

### LaserBeamRiderConfig()

初始化激光波束制导仪配置的新实例

#### Declaration

```
public LaserBeamRiderConfig()
```

#### Remarks

设置默认值：

- ID为空字符串
- 初始位置为原点(0,0,0)
- 激光功率为0瓦特
- 控制场直径为0米

# Properties

## ControlFieldDiameter

获取或设置控制场直径

Declaration

```
public double ControlFieldDiameter { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米 定义了导弹可以接收制导信号的横向范围

## Id

获取或设置激光波束制导仪的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的波束制导仪

## InitialPosition

获取或设置波束制导仪的初始位置

Declaration

```
public Vector3D InitialPosition { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示空间位置 坐标系：右手坐标系，X向右，Y向上，Z向前

## LaserPower

获取或设置激光功率

Declaration

```
public double LaserPower { get; set; }
```

Property Value



Type	Description
<a href="#">double</a>	

Remarks

单位：瓦特 影响制导波束的有效距离和导引精度

MaxGuidanceDistance

获取或设置最大导引距离

Declaration

```
public double MaxGuidanceDistance { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：米 定义了波束制导系统的最大有效工作距离

# Class LaserWarnerAlarmEvent

激光告警器警报事件，表示检测到激光照射

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserWarnerAlarmEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserWarnerAlarmEvent : SimulationEvent
```

## Remarks

用于模拟激光告警器的响应 触发时机：检测到激光照射时

## Properties

### TargetId

获取或设置被照射目标的ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识检测到激光照射的实体

[Show / Hide Table of Contents](#)

# Class LaserJammingEvent

激光干扰事件，表示对激光制导系统的干扰

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserJammingEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class LaserJammingEvent : SimulationEvent
```

## Remarks

用于模拟激光干扰效果 包含干扰功率信息

## Properties

### JammingPower

获取或设置干扰功率值

#### Declaration

```
public double JammingPower { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

#### Remarks

单位：瓦特 表示干扰源的输出功率

## TargetId

获取或设置被干扰目标的ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

标识受到激光干扰的目标实体

# Struct Vector2D

表示二维平面上的向量

## Inherited Members

[ValueType.Equals\(object\)](#)  
[ValueType.GetHashCode\(\)](#)  
[ValueType.ToString\(\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetType\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)

**Namespace:** [ThreatSource.Utils](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public struct Vector2D
```

## Remarks

用于处理二维平面上的计算，如目标投影等

## Constructors

### Vector2D(double, double)

初始化二维向量的新实例

## Declaration

```
public Vector2D(double x, double y)
```

## Parameters

Type	Name	Description
<a href="#">double</a>	<i>x</i>	X分量的值
<a href="#">double</a>	<i>y</i>	Y分量的值

## Properties

### X

获取或设置向量的X分量

## Declaration

```
public double X { readonly get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## Y

获取或设置向量的Y分量

Declaration

```
public double Y { readonly get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## Zero

零向量 (0, 0)

Declaration

```
public static Vector2D Zero { get; }
```

Property Value

Type	Description
<a href="#">Vector2D</a>	

# Interface ISimulationAdapter

第三方仿真环境适配器接口

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface ISimulationAdapter
```

## Remarks

该接口用于与外部仿真环境进行集成，提供实体查询和事件交互功能。实现此接口可以将威胁源仿真库与其他仿真系统对接。

## Methods

### GetEntity(string)

从外部仿真环境获取实体

Declaration

```
object? GetEntity(string id)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	实体ID

Returns

Type	Description
<a href="#">object</a>	实体对象，如果不存在则返回null

### PublishToExternalSimulation<T>(T)

发布事件到第三方仿真环境

Declaration

```
void PublishToExternalSimulation<T>(T evt)
```

Parameters

Type	Name	Description
T	<i>evt</i>	要发布的事件

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

用于将威胁源仿真库中的事件同步到外部仿真环境

ReceiveFromExternalSimulation<T>(T)

从第三方仿真环境接收事件

Declaration

```
void ReceiveFromExternalSimulation<T>(T evt)
```

Parameters

Type	Name	Description
T	<i>evt</i>	接收到的事件

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

用于接收外部仿真环境的事件并在威胁源仿真库中处理



[Show / Hide Table of Contents](#)

# Class LaserWarnerConfig

激光告警器配置类，用于设置激光探测和告警系统的参数

## Inheritance

↳ [object](#)  
↳ LaserWarnerConfig

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserWarnerConfig
```

## Remarks

该类定义了激光告警器的关键参数：

- 设备标识
- 灵敏度阈值
- 警报持续时间
- 探测波长范围 这些参数决定了告警器的探测能力和响应特性

## Constructors

### LaserWarnerConfig()

初始化激光告警器配置的新实例

## Declaration

```
public LaserWarnerConfig()
```

## Remarks

设置默认值：

- ID为空字符串
- 灵敏度阈值为0
- 警报持续时间为5秒
- 波长范围为0-0纳米

# Properties

## AlarmDuration

获取或设置警报持续时间

Declaration

```
public double AlarmDuration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 定义了每次告警的持续时间

## Id

获取或设置激光告警器的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的告警器

## SensitivityThreshold

获取或设置告警灵敏度阈值

Declaration

```
public double SensitivityThreshold { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特/平方米 当接收到的激光功率密度超过此阈值时触发告警

## WavelengthMax

获取或设置最大探测波长

Declaration

```
public double WavelengthMax { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最长波长

WavelengthMin

获取或设置最小探测波长

Declaration

```
public double WavelengthMin { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最短波长

# Class InfraredCommandGuidanceSystem

红外指令导引系统类，实现了基于红外跟踪器的指令制导功能

## Inheritance

↳ [object](#)  
↳ [BasicGuidanceSystem](#)  
↳ [InfraredCommandGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[BasicGuidanceSystem.HasGuidance](#)  
[BasicGuidanceSystem.MaxAcceleration](#)  
[BasicGuidanceSystem.ProportionalNavigationCoefficient](#)  
[BasicGuidanceSystem.Position](#)  
[BasicGuidanceSystem.Velocity](#)  
[BasicGuidanceSystem.GuidanceAcceleration](#)  
[BasicGuidanceSystem.GetGuidanceAcceleration\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredCommandGuidanceSystem : BasicGuidanceSystem, IGuidanceSystem
```

## Remarks

该类提供了红外指令制导系统的核心功能：

- 指令接收和处理
- 转向速率控制
- 提前量计算
- 制导加速度生成 用于实现地面指令制导的导弹控制

## Constructors

### InfraredCommandGuidanceSystem(double, double)

初始化红外指令导引系统的新实例

## Declaration

```
public InfraredCommandGuidanceSystem(double maxAcceleration, double guidanceCoefficient)
```

#### Parameters

Type	Name	Description
double	<i>maxAcceleration</i>	最大加速度，单位：米/平方秒
double	<i>guidanceCoefficient</i>	制导系数

#### Remarks

构造过程：

- 初始化基类参数
- 初始化向量记录
- 清零转向速率

## Methods

### CalculateGuidanceAcceleration(double)

计算制导加速度

#### Declaration

```
protected override void CalculateGuidanceAcceleration(double deltaTime)
```

#### Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间间隔，单位：秒

#### Overrides

[BasicGuidanceSystem.CalculateGuidanceAcceleration\(double\)](#)

#### Remarks

计算过程：

- 计算期望方向
- 更新转向速率
- 计算提前量
- 生成制导指令
- 限制最大加速度

### GetStatus()

获取制导系统的详细状态信息

#### Declaration

```
public override string GetStatus()
```

#### Returns

Type	Description
string	包含完整状态参数的字符串

#### Overrides

[BasicGuidanceSystem.GetStatus\(\)](#)

#### Remarks

返回信息：

- 基本状态信息
- 制导加速度
- 方向向量
- 转向速率 用于系统监控和调试

## ReceiveGuidanceCommand(Vector3D, Vector3D)

接收并处理制导指令

Declaration

```
public void ReceiveGuidanceCommand(Vector3D trackerToMissileVector, Vector3D trackerToTargetVector)
```

Parameters

Type	Name	Description
Vector3D	trackerToMissileVector	跟踪器到导弹的方向向量
Vector3D	trackerToTargetVector	跟踪器到目标的方向向量

Remarks

处理过程：

- 更新制导状态
- 记录方向向量
- 准备制导计算

## Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

Declaration

```
public override void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

Parameters

Type	Name	Description
double	deltaTime	自上次更新以来的时间间隔，单位：秒
Vector3D	missilePosition	导弹当前位置，单位：米
Vector3D	missileVelocity	导弹当前速度，单位：米/秒

Overrides

[BasicGuidanceSystem.Update\(double, Vector3D, Vector3D\)](#)

Remarks

更新过程：

- 更新基类状态
- 计算制导加速度

## Implements

[IGuidanceSystem](#)

# Class Sensor

传感器的抽象基类，实现了ISensor接口的基本功能

## Inheritance

↳ [object](#)

- ↳ [Sensor](#)
  - ↳ [InfraredDetector](#)
  - ↳ [LaserRangefinder](#)
  - ↳ [MillimeterWaveAltimeter](#)
  - ↳ [MillimeterWaveRadiometer](#)

## Implements

[ISensor](#)

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public abstract class Sensor : ISensor
```

## Remarks

该类提供了传感器的通用实现：

- 位置和姿态管理
- 激活状态控制
- 数据采集框架 是具体传感器类的实现基础

## Constructors

### Sensor(Vector3D, Orientation)

初始化传感器的新实例

## Declaration

```
protected Sensor(Vector3D position, Orientation orientation)
```

## Parameters

Type	Name	Description
------	------	-------------

Type	Name	Description
Vector3D	<i>position</i>	传感器的初始位置
Orientation	<i>orientation</i>	传感器的初始朝向

Remarks

构造过程：

- 设置初始位置
- 设置初始朝向
- 初始化工作状态

## Properties

### IsActive

获取或设置传感器是否处于激活状态

Declaration

```
public bool IsActive { get; set; }
```

Property Value

Type	Description
bool	

Remarks

true表示传感器正在工作 false表示传感器处于待机状态 控制传感器的工作模式

### Orientation

获取或设置传感器的朝向

Declaration

```
protected Orientation Orientation { get; set; }
```

Property Value

Type	Description
Orientation	

Remarks

传感器在三维空间中的姿态 影响传感器的探测方向和精度

### Position

获取或设置传感器的位置

Declaration

```
protected Vector3D Position { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks



传感器在三维空间中的位置向量 用于空间定位和测量计算

## Methods

### Activate()

激活传感器，使其开始工作

Declaration

```
public virtual void Activate()
```

Remarks

激活过程：

- 设置激活标志
- 准备数据采集
- 开始工作循环

### Deactivate()

停用传感器，使其停止工作

Declaration

```
public virtual void Deactivate()
```

Remarks

停用过程：

- 清除激活标志
- 停止数据采集
- 清理工作状态

### GetSensorData()

获取传感器数据（需要在子类中实现）

Declaration

```
public abstract SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含传感器测量结果的数据对象

Remarks

数据要求：

- 包含最新测量结果
- 附加时间戳信息
- 提供状态标志
- 确保数据有效性

### Update(double)

更新传感器状态（需要在子类中实现）

Declaration

```
public abstract void Update(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Remarks

更新要求：

- 检查工作状态
- 采集最新数据
- 更新内部状态
- 处理异常情况

## Implements

[ISensor](#)

# Class MotionAlgorithm

运动算法静态类，提供各种运动计算方法

## Inheritance

↳ [object](#)  
↳ MotionAlgorithm

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Utils](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public static class MotionAlgorithm
```

## Remarks

该类包含了导弹运动相关的各种算法，包括：

- 弹道计算
- 运动学计算
- 制导算法
- 扰动计算 所有方法都是静态的，可以直接调用。

## Methods

### AddRandomPerturbation(Vector3D)

为向量添加高斯噪声

#### Declaration

```
public static Vector3D AddRandomPerturbation(Vector3D vector)
```

#### Parameters

Type	Name	Description
<a href="#">Vector3D</a>	<i>vector</i>	原始向量

#### Returns

Type	Description
------	-------------

Type	Description
Vector3D	添加高斯噪声后的向量

Remarks

使用Box-Muller变换生成高斯随机数：

- 为向量的每个分量添加独立的高斯噪声
- 噪声强度由标准差控制（默认0.1）
- 用于模拟传感器误差和环境扰动

CalculateBallisticMotion(Vector3D, Vector3D, Vector3D, double)

使用运动学定律计算导弹运动状态

Declaration

```
public static (Vector3D newPosition, Vector3D newVelocity) CalculateBallisticMotion(Vector3D currentPosition, Vector3D currentVelocity, Vector3D acceleration, double deltaTime)
```

Parameters

Type	Name	Description
Vector3D	currentPosition	当前位置坐标
Vector3D	currentVelocity	当前速度向量
Vector3D	acceleration	加速度向量（包含重力加速度）
double	deltaTime	时间步长，单位：秒

Returns

Type	Description
(Vector3D newPosition, Vector3D newVelocity)	包含新位置和新速度的元组

Remarks

该方法适用于无制导状态下的导弹运动计算，使用标准运动学方程：

- 位置更新：  $p = p_0 + v_0t + 0.5a*t^2$
- 速度更新：  $v = v_0 + a*t$

CalculateBestLaunchOrientation(Vector3D, Vector3D, double)

计算抛物线弹道最佳发射方向（选择较小的仰角）

Declaration

```
public static (Orientation? orientation, Vector3D? velocity) CalculateBestLaunchOrientation(Vector3D startPos, Vector3D targetPos, double initialSpeed)
```

Parameters

Type	Name	Description
Vector3D	startPos	发射位置坐标
Vector3D	targetPos	目标位置坐标
double	initialSpeed	发射初速度，单位：米/秒

Returns

Type	Description
(Orientation? orientation, Vector3D velocity)	包含最佳发射方向和初始速度向量的元组，若无解则返回null

Remarks

该方法使用弹道方程计算两个可能的发射角度，并选择较小的仰角作为最佳发射方向。 计算考虑了重力影响，但未考虑空气阻力。

## CalculateLaunchAngles(double, double, double, double)

计算抛物线弹道发射角度

Declaration

```
public static double[]? CalculateLaunchAngles(double v0, double x, double y, double g = 9.81)
```

Parameters

Type	Name	Description
double	<i>v0</i>	初始速度，单位：米/秒
double	<i>x</i>	目标水平距离，单位：米
double	<i>y</i>	目标高度差，单位：米
double	<i>g</i>	重力加速度，默认9.81米/秒²

Returns

Type	Description
double[]	两个可能的发射角度（弧度），如果无解则返回null

Remarks

使用标准弹道方程计算发射角度，会返回两个解：

- 一个是低角度解（较小仰角）
- 一个是高角度解（较大仰角） 如果目标距离超出武器射程，则返回null。

## CalculateProportionalNavigation(double, Vector3D, Vector3D, Vector3D, Vector3D)

计算比例导引加速度

Declaration

```
public static Vector3D CalculateProportionalNavigation(double proportionalNavigationCoefficient, Vector3D missilePosition, Vector3D missileVelocity, Vector3D targetPosition, Vector3D targetVelocity)
```

Parameters

Type	Name	Description
double	<i>proportionalNavigationCoefficient</i>	比例导引系数
Vector3D	<i>missilePosition</i>	导弹当前位置
Vector3D	<i>missileVelocity</i>	导弹当前速度
Vector3D	<i>targetPosition</i>	目标当前位置
Vector3D	<i>targetVelocity</i>	目标当前速度

Returns

Type	Description
Vector3D	比例导引产生的加速度向量

Remarks

使用比例导引法计算制导加速度：

- 预测目标未来位置
- 计算视线角速率

- 根据比例导引公式计算所需加速度 加速度方向垂直于导弹速度方向。

## RungeKutta4(double, Vector3D, Vector3D, Vector3D)

使用四阶龙格库塔法计算导弹运动状态

### Declaration

```
public static (Vector3D newPosition, Vector3D newVelocity) RungeKutta4(double deltaTime,
    Vector3D position, Vector3D velocity, Vector3D acceleration)
```

### Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒
Vector3D	<i>position</i>	当前位置坐标
Vector3D	<i>velocity</i>	当前速度向量
Vector3D	<i>acceleration</i>	当前加速度向量

### Returns

Type	Description
(Vector3D newPosition, Vector3D newVelocity)	包含新位置和新速度的元组

### Remarks

四阶龙格库塔法提供了更高精度的数值解：

- 适用于有制导状态下的导弹运动计算
- 考虑了加速度随时间的变化
- 比简单欧拉法具有更高的精度

[Show / Hide Table of Contents](#)

# Class EntityDestroyedEvent

实体销毁事件，表示仿真实体被销毁

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ EntityDestroyedEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class EntityDestroyedEvent : SimulationEvent
```

## Remarks

用于通知系统某个实体已被销毁 触发时机：实体被摧毁或删除时

## Properties

### DestroyedEntityId

获取或设置被销毁实体的ID

#### Declaration

```
public string? DestroyedEntityId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识已被销毁的实体

[Show / Hide Table of Contents](#)

# Class MillimeterWaveJammingEvent

毫米波干扰事件，表示对毫米波雷达的干扰

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [MillimeterWaveJammingEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveJammingEvent : SimulationEvent
```

## Remarks

用于模拟对毫米波制导系统的干扰 包含干扰功率信息

## Properties

### JammingPower

获取或设置干扰功率值

#### Declaration

```
public double JammingPower { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

#### Remarks

单位：瓦特 表示毫米波干扰源的输出功率



# TargetId

获取或设置被干扰目标的ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

标识受到毫米波干扰的目标实体

# Namespace ThreatSource.Missile

## Classes

### BaseMissile

导弹基类，实现了导弹的基本功能和状态管理

### InfraredCommandGuidedMissile

红外指令制导导弹类，实现了红外指令制导的导弹功能

### InfraredImagingTerminalGuidedMissile

红外成像末制导导弹类，继承自基础导弹类

### LaserBeamRiderMissile

激光波束制导导弹类，实现了激光波束跟踪制导的导弹功能

### LaserSemiActiveGuidedMissile

激光半主动制导导弹类，实现了激光半主动寻的制导功能

### MillimeterWaveTerminalGuidedMissile

毫米波末制导导弹类，继承自基础导弹类

### MissileProperties

导弹配置类，定义了导弹的所有基本属性和性能参数

### TerminalSensitiveMissile

末敏导弹类，实现了末端敏感引信的导弹功能

### TerminalSensitiveSubmunition

末敏子弹类，实现了多传感器融合的末端制导功能

## Structs

### MissileRunningState

导弹运行状态信息结构体，包含导弹的完整状态数据

## Interfaces

## IMissile

导弹接口，定义了导弹的基本行为和状态

## Enums

## MissileType

导弹类型枚举，定义了系统支持的所有导弹类型

[Show / Hide Table of Contents](#)

# Class LaserDesignatorConfig

激光指示器配置类，用于设置激光半主动导引系统中的激光指示器参数

## Inheritance

↳ [object](#)

↳ LaserDesignatorConfig

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserDesignatorConfig
```

## Remarks

该类定义了激光指示器的关键参数：

- 设备标识
- 空间位置
- 激光功率
- 光束发散角 这些参数影响激光指示器的性能和工作特性

## Constructors

### LaserDesignatorConfig()

初始化激光指示器配置的新实例

## Declaration

```
public LaserDesignatorConfig()
```

## Remarks

设置默认值：

- ID为空字符串
- 初始位置为原点(0,0,0)
- 激光功率为0瓦特
- 发散角为0弧度

# Properties

## Id

获取或设置激光指示器的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的激光指示器

## InitialPosition

获取或设置激光指示器的初始位置

Declaration

```
public Vector3D InitialPosition { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示空间位置 坐标系：右手坐标系，X向右，Y向上，Z向前

## LaserDivergenceAngle

获取或设置激光发散角

Declaration

```
public double LaserDivergenceAngle { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：弧度 影响激光束的扩散特性和照射面积

## LaserPower

获取或设置激光功率

Declaration

```
public double LaserPower { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特 影响激光照射的有效距离和目标反射强度

# Class RangefinderSensorData

测距仪传感器数据类，包含距离测量的具体数据

## Inheritance

↳ [object](#)  
↳ [SensorData](#)  
↳ [RangefinderSensorData](#)

## Inherited Members

[SensorData.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class RangefinderSensorData : SensorData
```

## Remarks

该类封装了激光测距仪的测量结果：

- 目标距离信息 用于目标定位和距离测量

# Properties

## Distance

获取或设置测量的距离，单位：米

## Declaration

```
public double Distance { get; set; }
```

## Property Value

Type	Description
<a href="#">double</a>	

## Remarks

# Struct Orientation

表示三维空间中的方向，使用欧拉角表示

## Inherited Members

[ValueType.Equals\(object\)](#)  
[ValueType.GetHashCode\(\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetType\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)

**Namespace:** [ThreatSource.Utils](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public struct Orientation
```

## Remarks

使用欧拉角（偏航角、俯仰角、滚转角）来表示三维空间中的方向：

- 偏航角（Yaw）：绕Y轴旋转的角度
- 俯仰角（Pitch）：绕X轴旋转的角度
- 滚转角（Roll）：绕Z轴旋转的角度 所有角度均使用弧度制

## Constructors

### Orientation(double, double, double)

初始化方向的新实例

## Declaration

```
public Orientation(double yaw, double pitch, double roll)
```

## Parameters

Type	Name	Description
<a href="#">double</a>	<i>yaw</i>	偏航角，单位：弧度
<a href="#">double</a>	<i>pitch</i>	俯仰角，单位：弧度
<a href="#">double</a>	<i>roll</i>	滚转角，单位：弧度

## Properties

### Pitch



获取或设置俯仰角（绕X轴旋转的角度），单位：弧度

Declaration

```
public double Pitch { readonly get; set; }
```

Property Value

Type	Description
double	

Roll

获取或设置滚转角（绕Z轴旋转的角度），单位：弧度

Declaration

```
public double Roll { readonly get; set; }
```

Property Value

Type	Description
double	

Yaw

获取或设置偏航角（绕Y轴旋转的角度），单位：弧度

Declaration

```
public double Yaw { readonly get; set; }
```

Property Value

Type	Description
double	

Methods

FromVector(Vector3D)

从向量创建方向

Declaration

```
public static Orientation FromVector(Vector3D vector)
```

Parameters

Type	Name	Description
Vector3D	<i>vector</i>	输入向量

Returns

Type	Description
Orientation	对应的方向

## LookAt(Vector3D)

根据给定的方向向量创建方向

Declaration

```
public static Orientation LookAt(Vector3D direction)
```

Parameters

Type	Name	Description
<a href="#">Vector3D</a>	<i>direction</i>	方向向量

Returns

Type	Description
<a href="#">Orientation</a>	对应的方向

## Normalize()

将角度归一化到  $[-\pi, \pi]$  范围内

Declaration

```
public void Normalize()
```

## ToString()

将方向转换为字符串表示

Declaration

```
public override readonly string ToString()
```

Returns

Type	Description
<a href="#">string</a>	方向的字符串表示

Overrides

[ValueType.ToString\(\)](#)

## ToVector()

将方向转换为单位向量

Declaration

```
public readonly Vector3D ToVector()
```

Returns

Type	Description
<a href="#">Vector3D</a>	对应的单位向量

# Class LaserBeamStartEvent

激光波束开始事件，表示激光波束制导开始

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserBeamStartEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserBeamStartEvent : SimulationEvent
```

## Remarks

用于激光波束制导系统 触发时机：开始发射制导激光波束时

## Properties

### LaserBeamRiderId

获取或设置激光波束制导器的ID

#### Declaration

```
public string? LaserBeamRiderId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

[Show / Hide Table of Contents](#)

# Class UltravioletDetectionEvent

紫外探测事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ UltravioletDetectionEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class UltravioletDetectionEvent : SimulationEvent
```

## Properties

### Intensity

紫外辐射强度

#### Declaration

```
public double Intensity { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

威胁源仿真库文档

[Back to top](#)

# Class LaserRangefinder

激光测距仪类，实现了目标距离测量和数据采集功能

## Inheritance

↳ [object](#)  
↳ [Sensor](#)  
↳ [LaserRangefinder](#)

## Implements

[ISensor](#)

## Inherited Members

[Sensor.IsActive](#)  
[Sensor.Position](#)  
[Sensor.Orientation](#)  
[Sensor.Activate\(\)](#)  
[Sensor.Deactivate\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserRangefinder : Sensor, ISensor
```

## Remarks

该类提供了激光测距仪的核心功能：

- 实时距离测量
- 脉冲频率控制
- 量程范围控制
- 数据采集 用于目标距离的精确测量和跟踪

## Constructors

### LaserRangefinder(Vector3D, Orientation, double, double)

初始化激光测距仪的新实例

## Declaration

```
public LaserRangefinder(Vector3D position, Orientation orientation, double maxRange, double pulseRate)
```

Parameters

Type	Name	Description
Vector3D	<i>position</i>	测距仪的位置坐标
Orientation	<i>orientation</i>	测距仪的朝向
double	<i>maxRange</i>	最大测量距离，单位：米
double	<i>pulseRate</i>	脉冲频率，单位：赫兹

Remarks

构造过程：

- 设置位置和姿态
- 配置测量参数
- 初始化工作状态

## Properties

### MaxRange

获取或设置最大测量距离，单位：米

Declaration

```
public double MaxRange { get; set; }
```

Property Value

Type	Description
double	

Remarks

定义了测距仪的量程上限 超出此距离的测量可能不准确 受大气条件和目标反射率影响

### PulseRate

获取或设置脉冲频率，单位：赫兹

Declaration

```
public double PulseRate { get; set; }
```

Property Value

Type	Description
double	

Remarks

定义了激光发射的重复频率 影响测量的更新速率和精度 需要根据目标运动特性选择

## Methods

### GetSensorData()

获取激光测距仪的最新数据

Declaration

```
public override SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含距离测量结果的数据对象

Overrides

[Sensor.GetSensorData\(\)](#)

Remarks

返回数据：

- 目标距离值
- 信号强度
- 测量时间戳
- 数据可靠性

## Update(double)

更新激光测距仪的工作状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Overrides

[Sensor.Update\(double\)](#)

Remarks

更新过程：

- 发射激光脉冲
- 接收回波信号
- 计算目标距离
- 更新测量数据

## Implements

[ISensor](#)



[Show / Hide Table of Contents](#)

# Enum IndicatorType

指示器类型枚举，定义了支持的指示器类型

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

Syntax

```
public enum IndicatorType
```

## Remarks

包含三种主要的指示器类型：

- LaserDisintegrator：激光指示器，用于半主动激光制导
- LaserBeamRider：激光波束制导器，用于波束乘波制导
- InfraredTracker：红外跟踪器，用于红外制导 每种类型具有不同的工作原理和性能特点

## Fields

Name	Description
InfraredTracker	
LaserBeamRider	
LaserDisintegrator	

# Class MillimeterWaveWarnerConfig

毫米波告警器配置类，用于设置毫米波探测和告警系统的参数

## Inheritance

↳ [object](#)

↳ MillimeterWaveWarnerConfig

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveWarnerConfig
```

## Remarks

该类定义了毫米波告警器的关键参数：

- 设备标识
- 灵敏度阈值
- 警报持续时间
- 探测波长范围 这些参数决定了告警器对毫米波辐射的探测能力

## Constructors

### MillimeterWaveWarnerConfig()

初始化毫米波告警器配置的新实例

## Declaration

```
public MillimeterWaveWarnerConfig()
```

## Remarks

设置默认值：

- ID为空字符串
- 灵敏度阈值为0
- 警报持续时间为5秒
- 波长范围为0-0纳米

# Properties

## AlarmDuration

获取或设置警报持续时间

Declaration

```
public double AlarmDuration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 定义了每次告警的持续时间

## Id

获取或设置毫米波告警器的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的告警器

## SensitivityThreshold

获取或设置告警灵敏度阈值

Declaration

```
public double SensitivityThreshold { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特/平方米 当接收到的毫米波辐射强度超过此阈值时触发告警

## WavelengthMax

获取或设置最大探测波长

Declaration

```
public double WavelengthMax { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最长毫米波波长

WavelengthMin

获取或设置最小探测波长

Declaration

```
public double WavelengthMin { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最短毫米波波长

# Class LaserSemiActiveGuidanceSystem

激光半主动制导系统类，实现了基于激光照射的目标跟踪和制导功能

## Inheritance

↳ [object](#)  
↳ [BasicGuidanceSystem](#)  
↳ [LaserSemiActiveGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[BasicGuidanceSystem.HasGuidance](#)  
[BasicGuidanceSystem.MaxAcceleration](#)  
[BasicGuidanceSystem.ProportionalNavigationCoefficient](#)  
[BasicGuidanceSystem.Position](#)  
[BasicGuidanceSystem.Velocity](#)  
[BasicGuidanceSystem.GuidanceAcceleration](#)  
[BasicGuidanceSystem.GetGuidanceAcceleration\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserSemiActiveGuidanceSystem : BasicGuidanceSystem, IGuidanceSystem
```

## Remarks

该类提供了激光半主动制导系统的核心功能：

- 激光目标照射
- 反射光探测
- 信号处理
- 比例导引控制 用于实现精确制导打击

## Constructors

### LaserSemiActiveGuidanceSystem(double, double)

初始化激光半主动制导系统的新实例

## Declaration

```
public LaserSemiActiveGuidanceSystem(double maxAcceleration, double guidanceCoefficient)
```

Parameters

Type	Name	Description
double	<i>maxAcceleration</i>	最大加速度，单位：米/平方秒
double	<i>guidanceCoefficient</i>	制导系数

Remarks

构造过程：

- 初始化基类参数
- 初始化目标信息
- 初始化激光参数

## Methods

### CalculateGuidanceAcceleration(double)

计算制导加速度

Declaration

```
protected override void CalculateGuidanceAcceleration(double deltaTime)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	时间步长，单位：秒

Overrides

[BasicGuidanceSystem.CalculateGuidanceAcceleration\(double\)](#)

Remarks

计算过程：

- 使用比例导引算法
- 考虑目标运动
- 限制最大加速度

### DeactivateLaserDesignator()

关闭激光照射系统

Declaration

```
public void DeactivateLaserDesignator()
```

Remarks

关闭过程：

- 停止激光照射
- 清除位置信息
- 清除目标信息
- 清除激光参数

### GetStatus()

获取制导系统的详细状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含完整状态参数的字符串

Overrides

BasicGuidanceSystem.GetStatus()

Remarks

返回信息：

- 基本状态信息
- 接收功率数据
- 锁定阈值 用于系统监控和调试

Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

Declaration

```
public override void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

Parameters

Type	Name	Description
double	deltaTime	时间步长，单位：秒
Vector3D	missilePosition	导弹位置，单位：米
Vector3D	missileVelocity	导弹速度，单位：米/秒

Overrides

BasicGuidanceSystem.Update(double, Vector3D, Vector3D)

Remarks

更新过程：

- 检查激光照射状态
- 计算接收功率
- 判断是否锁定目标
- 计算制导指令

UpdateLaserDesignator(Vector3D, Vector3D, Vector3D, double, double)

更新激光指示器参数

Declaration

```
public void UpdateLaserDesignator(Vector3D sourcePosition, Vector3D targetPosition, Vector3D targetVelocity, double laserPower, double laserDivergenceAngle)
```

Parameters

Type	Name	Description
Vector3D	sourcePosition	激光源位置，单位：米
Vector3D	targetPosition	目标位置，单位：米
Vector3D	targetVelocity	目标速度，单位：米/秒

Type	Name	Description
<a href="#">double</a>	<i>laserPower</i>	激光功率，单位：瓦特
<a href="#">double</a>	<i>laserDivergenceAngle</i>	激光发散角，单位：弧度

Remarks

更新过程：

- 激活激光照射
- 更新位置信息
- 更新目标信息
- 更新激光参数

## Implements

[IGuidanceSystem](#)



[Show / Hide Table of Contents](#)

# Enum IndicatorState

指示器状态枚举，定义了指示器的工作状态

**Namespace:** [ThreatSource.Indicator](#)

**Assembly:** ThreatSource.dll

Syntax

```
public enum IndicatorState
```

## Remarks

包含两种基本状态：

- Indicating：指示状态，正在执行指示或跟踪任务
- Idle：空闲状态，等待新的指示任务 用于控制指示器的工作流程

## Fields

Name	Description
Idle	
Indicating	

# Class Vector3D

表示三维空间中的向量，提供基本的向量运算功能

## Inheritance

↳ [object](#)  
↳ Vector3D

## Inherited Members

[object.Equals\(object, object\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)

**Namespace:** [ThreatSource.Utils](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class Vector3D
```

## Remarks

该类实现了三维向量的基本运算，包括：

- 向量的加减乘除运算
- 向量的点积和叉积
- 向量的归一化
- 向量的旋转变换 所有计算都使用双精度浮点数（double）以保证精度

## Constructors

### Vector3D(double, double, double)

初始化三维向量的新实例

## Declaration

```
public Vector3D(double x, double y, double z)
```

## Parameters

Type	Name	Description
<a href="#">double</a>	<i>x</i>	X分量的值
<a href="#">double</a>	<i>y</i>	Y分量的值
<a href="#">double</a>	<i>z</i>	Z分量的值

## Properties

### UnitX

X轴单位向量 (1, 0, 0)

Declaration

```
public static Vector3D UnitX { get; }
```

Property Value

Type	Description
Vector3D	

### UnitY

Y轴单位向量 (0, 1, 0)

Declaration

```
public static Vector3D UnitY { get; }
```

Property Value

Type	Description
Vector3D	

### UnitZ

Z轴单位向量 (0, 0, 1)

Declaration

```
public static Vector3D UnitZ { get; }
```

Property Value

Type	Description
Vector3D	

### X

获取或设置向量的X分量

Declaration

```
public double X { get; set; }
```

Property Value

Type	Description
double	

### Y

获取或设置向量的Y分量

Declaration

```
public double Y { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## Z

获取或设置向量的Z分量

Declaration

```
public double Z { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## Zero

零向量 (0, 0, 0)

Declaration

```
public static Vector3D Zero { get; }
```

Property Value

Type	Description
<a href="#">Vector3D</a>	

## Methods

### CrossProduct(Vector3D, Vector3D)

计算两个向量的叉积

Declaration

```
public static Vector3D CrossProduct(Vector3D a, Vector3D b)
```

Parameters

Type	Name	Description
<a href="#">Vector3D</a>	<i>a</i>	第一个向量
<a href="#">Vector3D</a>	<i>b</i>	第二个向量

Returns

Type	Description
<a href="#">Vector3D</a>	叉积结果

### Distance(Vector3D, Vector3D)

计算两个向量之间的距离

Declaration

```
public static double Distance(Vector3D v1, Vector3D v2)
```

Parameters

Type	Name	Description
Vector3D	v1	第一个向量
Vector3D	v2	第二个向量

Returns

Type	Description
double	两个向量之间的距离

DotProduct(Vector3D, Vector3D)

计算两个向量的点积

Declaration

```
public static double DotProduct(Vector3D a, Vector3D b)
```

Parameters

Type	Name	Description
Vector3D	a	第一个向量
Vector3D	b	第二个向量

Returns

Type	Description
double	点积结果

Equals(object?)

判断两个向量是否相等

Declaration

```
public override bool Equals(object? obj)
```

Parameters

Type	Name	Description
object	obj	

Returns

Type	Description
bool	

Overrides

object.Equals(object)

GetHashCode()

获取向量的哈希码

Declaration

```
public override int GetHashCode()
```

Returns

Type	Description
int	向量的哈希码

Overrides

[object.GetHashCode\(\)](#)

Magnitude()

计算向量的模长

Declaration

```
public double Magnitude()
```

Returns

Type	Description
double	向量的模长

MagnitudeSquared()

计算向量模长的平方

Declaration

```
public double MagnitudeSquared()
```

Returns

Type	Description
double	向量模长的平方

Negate(Vector3D)

向量取反

Declaration

```
public static Vector3D Negate(Vector3D a)
```

Parameters

Type	Name	Description
<a href="#">Vector3D</a>	<i>a</i>	输入向量

Returns

Type	Description
<a href="#">Vector3D</a>	取反后的向量

Normalize()

向量归一化

Declaration

```
public Vector3D Normalize()
```

Returns

Type	Description
Vector3D	归一化后的向量

PointOnLine(Vector3D, Vector3D, double)

计算AB两点之间距离 A 点一定距离的点的坐标

Declaration

```
public static Vector3D PointOnLine(Vector3D a, Vector3D b, double distance)
```

Parameters

Type	Name	Description
Vector3D	<i>a</i>	直线起点
Vector3D	<i>b</i>	直线终点
double	<i>distance</i>	距离起点距离

Returns

Type	Description
Vector3D	直线上的点

ToString()

将向量转换为字符串表示

Declaration

```
public override string ToString()
```

Returns

Type	Description
string	向量的字符串表示

Overrides

[object.ToString\(\)](#)

Operators

operator +(Vector3D, Vector3D)

向量加法运算符重载

Declaration

```
public static Vector3D operator +(Vector3D a, Vector3D b)
```

Parameters

Type	Name	Description
------	------	-------------

Type	Name	Description
Vector3D	<i>a</i>	
Vector3D	<i>b</i>	

Returns

Type	Description
Vector3D	

## operator /(Vector3D, double)

向量与标量除法运算符重载

Declaration

```
public static Vector3D operator /(Vector3D a, double scalar)
```

Parameters

Type	Name	Description
Vector3D	<i>a</i>	
double	<i>scalar</i>	

Returns

Type	Description
Vector3D	

## operator ==(Vector3D, Vector3D)

向量相等运算符重载

Declaration

```
public static bool operator ==(Vector3D left, Vector3D right)
```

Parameters

Type	Name	Description
Vector3D	<i>left</i>	
Vector3D	<i>right</i>	

Returns

Type	Description
bool	

## operator !=(Vector3D, Vector3D)

向量不相等运算符重载

Declaration

```
public static bool operator !=(Vector3D left, Vector3D right)
```

Parameters

Type	Name	Description
------	------	-------------



Type	Name	Description
Vector3D	<i>left</i>	
Vector3D	<i>right</i>	

Returns

Type	Description
bool	

## operator \*(Vector3D, double)

向量与标量乘法运算符重载

Declaration

```
public static Vector3D operator *(Vector3D a, double scalar)
```

Parameters

Type	Name	Description
Vector3D	<i>a</i>	
double	<i>scalar</i>	

Returns

Type	Description
Vector3D	

## operator -(Vector3D, Vector3D)

向量减法运算符重载

Declaration

```
public static Vector3D operator -(Vector3D a, Vector3D b)
```

Parameters

Type	Name	Description
Vector3D	<i>a</i>	
Vector3D	<i>b</i>	

Returns

Type	Description
Vector3D	

## operator -(Vector3D)

向量反向

Declaration

```
public static Vector3D operator -(Vector3D a)
```

Parameters

Type	Name	Description
------	------	-------------

Type	Name	Description
<a href="#">Vector3D</a>	<i>a</i>	

Returns

Type	Description
<a href="#">Vector3D</a>	

# Class InfraredWarnerConfig

红外告警器配置类，用于设置红外探测和告警系统的参数

## Inheritance

↳ [object](#)  
↳ InfraredWarnerConfig

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredWarnerConfig
```

## Remarks

该类定义了红外告警器的关键参数：

- 设备标识
- 灵敏度阈值
- 警报持续时间
- 探测波长范围 这些参数决定了告警器对红外辐射的探测能力

## Constructors

### InfraredWarnerConfig()

初始化红外告警器配置的新实例

## Declaration

```
public InfraredWarnerConfig()
```

## Remarks

设置默认值：

- ID为空字符串
- 灵敏度阈值为0
- 警报持续时间为5秒
- 波长范围为0-0纳米

# Properties

## AlarmDuration

获取或设置警报持续时间

Declaration

```
public double AlarmDuration { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 定义了每次告警的持续时间

## Id

获取或设置红外告警器的唯一标识符

Declaration

```
public string Id { get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和查找特定的告警器

## SensitivityThreshold

获取或设置告警灵敏度阈值

Declaration

```
public double SensitivityThreshold { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：瓦特/平方米 当接收到的红外辐射强度超过此阈值时触发告警

## WavelengthMax

获取或设置最大探测波长

Declaration

```
public double WavelengthMax { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最长红外波长

WavelengthMin

获取或设置最小探测波长

Declaration

```
public double WavelengthMin { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

Remarks

单位：纳米 定义了告警器可以探测的最短红外波长

# Class EntityActivatedEvent

实体激活事件，表示仿真实体被激活

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [EntityActivatedEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class EntityActivatedEvent : SimulationEvent
```

## Remarks

用于通知系统某个实体已进入活动状态 触发时机：实体被创建或重新激活时

## Properties

### ActivatedEntityId

获取或设置被激活实体的ID

#### Declaration

```
public string? ActivatedEntityId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识已被激活的实体

# Class MillimeterWaveRadiometer

毫米波辐射计类，实现了目标辐射温度测量和目标识别功能

## Inheritance

↳ [object](#)  
↳ [Sensor](#)  
↳ [MillimeterWaveRadiometer](#)

## Implements

[ISensor](#)

## Inherited Members

[Sensor.IsActive](#)  
[Sensor.Position](#)  
[Sensor.Orientation](#)  
[Sensor.Activate\(\)](#)  
[Sensor.Deactivate\(\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveRadiometer : Sensor, ISensor
```

## Remarks

该类提供了毫米波辐射计的核心功能：

- 辐射温度测量
- 温差阈值检测
- 目标识别判断
- 实时状态更新 用于末敏子弹的目标探测和识别

## Constructors

### MillimeterWaveRadiometer(TerminalSensitiveSubmunition, double, double)

初始化毫米波辐射计的新实例

## Declaration

```
public MillimeterWaveRadiometer(TerminalSensitiveSubmunition submunition, double wavelength, double fieldOfView)
```

Parameters

Type	Name	Description
TerminalSensitiveSubmunition	<i>submunition</i>	末敏子弹实例
double	<i>wavelength</i>	工作波段，单位：毫米
double	<i>fieldOfView</i>	视场角，单位：度

Remarks

构造过程：

- 设置工作参数
- 初始化温度记录
- 创建传感器数据
- 继承基类位置和姿态

## Properties

### DetectionTemperatureDifferenceThreshold

获取或设置辐射温度差检测阈值，单位：开尔文

Declaration

```
public double DetectionTemperatureDifferenceThreshold { get; set; }
```

Property Value

Type	Description
double	

Remarks

辐射计高温物体与低温物体的检测温差 默认值为100K 典型温差值：

- 金属目标与草地：170~230K
- 金属目标与砂石地：120~150K 辐射率说明：
- 草地辐射率：1.0
- 砂石地辐射率：0.83
- 金属辐射率：0 坦克辐射温度等于天空背景辐射的反射温度，天顶角30度时约为90K

### Wavelength

获取或设置工作波段，单位：毫米

Declaration

```
public double Wavelength { get; set; }
```

Property Value

Type	Description
double	

Remarks

可选波段：3mm 或 8mm 影响辐射计的探测性能和大气衰减



# Methods

## GetSensorData()

获取毫米波辐射计的最新数据

Declaration

```
public override SensorData GetSensorData()
```

Returns

Type	Description
<a href="#">SensorData</a>	包含探测结果的数据对象

Overrides

[Sensor.GetSensorData\(\)](#)

Remarks

返回数据：

- 目标探测状态
- 温度测量结果
- 时间戳信息

## Update(double)

更新毫米波辐射计的工作状态

Declaration

```
public override void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒

Overrides

[Sensor.Update\(double\)](#)

Remarks

更新过程：

- 检查激活状态
- 测量辐射温度
- 计算温度差值
- 判断目标存在
- 更新历史温度

## Implements

[ISensor](#)

# Class LaserIlluminationStartEvent

激光照射开始事件，表示激光定位器开始照射目标

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserIlluminationStartEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserIlluminationStartEvent : SimulationEvent
```

## Remarks

用于激光半主动导引系统 触发时机：激光定位器开始照射目标时

# Properties

## LaserDesignatorId

获取或设置激光定位器的ID

## Declaration

```
public string? LaserDesignatorId { get; set; }
```

## Property Value

Type	Description
<a href="#">string</a>	

## Remarks

标识发出激光的定位器设备

# TargetId

获取或设置被照射目标的ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

标识被激光照射的目标实体

[Show / Hide Table of Contents](#)

# Class TankRadiationEvent

坦克辐射事件，表示坦克的多波段辐射特性

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ TankRadiationEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class TankRadiationEvent : SimulationEvent
```

## Remarks

用于模拟坦克在不同波段的辐射特征 包含激光、毫米波和红外波段的信息

## Properties

### InfraredRadiationIntensity

获取或设置红外辐射强度

#### Declaration

```
public double InfraredRadiationIntensity { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

#### Remarks

单位：瓦特/平方米 表示坦克的红外辐射强度

## LaserReflectionIntensity

获取或设置激光漫反射强度

Declaration

```
public double LaserReflectionIntensity { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：相对单位 表示坦克表面对激光的反射能力

## MillimeterWaveRadiationTemperature

获取或设置毫米波辐射温度

Declaration

```
public double MillimeterWaveRadiationTemperature { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：开尔文 表示坦克在毫米波波段的辐射温度

## MillimeterWaveReflectionIntensity

获取或设置毫米波反射强度

Declaration

```
public double MillimeterWaveReflectionIntensity { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：相对单位 表示坦克对毫米波雷达的反射特性

[Show / Hide Table of Contents](#)

# Class InfraredJammingEvent

红外干扰事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [InfraredJammingEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class InfraredJammingEvent : SimulationEvent
```

## Properties

### JammingPower

干扰功率(瓦特)

#### Declaration

```
public double JammingPower { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

## WavelengthMax

最大波长(微米)

Declaration

```
public double WavelengthMax { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

## WavelengthMin

最小波长(微米)

Declaration

```
public double WavelengthMin { get; set; }
```

Property Value

Type	Description
<a href="#">double</a>	

# Class InfraredSensorData

红外传感器数据类，包含红外探测的具体数据

## Inheritance

↳ [object](#)  
↳ [SensorData](#)  
↳ InfraredSensorData

## Inherited Members

[SensorData.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredSensorData : SensorData
```

## Remarks

该类封装了红外传感器的测量结果：

- 目标温度信息
- 目标探测状态 用于红外目标的探测和跟踪

# Properties

## IsTargetDetected

获取或设置是否检测到目标

## Declaration

```
public bool IsTargetDetected { get; set; }
```

## Property Value

Type	Description
<a href="#">bool</a>	

## Remarks

true表示探测到有效目标 false表示未探测到目标 用于目标存在性判断



# Temperature

获取或设置探测到的温度，单位：开尔文

## Declaration

```
public double Temperature { get; set; }
```

## Property Value

Type	Description
double	

## Remarks

记录目标的红外辐射温度 用于目标特征识别和状态评估

# Struct MissileRunningState

导弹运行状态信息结构体，包含导弹的完整状态数据

## Inherited Members

[ValueType.Equals\(object\)](#)  
[ValueType.GetHashCode\(\)](#)  
[ValueType.ToString\(\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetType\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public struct MissileRunningState
```

## Remarks

该结构体用于：

- 状态监控和记录
- 数据分析和诊断
- 仿真状态同步
- 导弹行为控制

## Properties

### EngineBurnTime

获取或设置发动机的当前燃烧时间

#### Declaration

```
public double EngineBurnTime { readonly get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

#### Remarks

单位：秒 记录发动机工作的累计时间 用于控制推力变化和燃料消耗

### FlightDistance

获取或设置导弹的当前飞行距离

Declaration

```
public double FlightDistance { readonly get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米 从发射点到当前位置的累计飞行距离

FlightTime

获取或设置导弹的当前飞行时间

Declaration

```
public double FlightTime { readonly get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 从发射时刻开始计时

Id

获取或设置导弹的唯一标识符

Declaration

```
public string Id { readonly get; set; }
```

Property Value

Type	Description
string	

Remarks

在仿真系统中必须唯一 用于标识和追踪特定的导弹

IsActive

获取或设置导弹是否处于活动状态

Declaration

```
public bool IsActive { readonly get; set; }
```

Property Value

Type	Description
bool	

Remarks

true表示导弹正常工作 false表示导弹已停止工作（销毁或自毁）

## IsGuidance

获取或设置导弹是否处于制导状态

Declaration

```
public bool IsGuidance { readonly get; set; }
```

Property Value

Type	Description
bool	

Remarks

true表示导弹当前在制导 false表示导弹处于非制导状态

## LostGuidanceTime

获取或设置导弹失去制导的持续时间

Declaration

```
public double LostGuidanceTime { readonly get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：秒 记录导弹处于非制导状态的累计时间 用于判断是否需要触发自毁

## Orientation

获取或设置导弹的当前朝向

Declaration

```
public Orientation Orientation { readonly get; set; }
```

Property Value

Type	Description
Orientation	

Remarks

使用欧拉角表示导弹的姿态 包含偏航角、俯仰角和滚转角

## Position

获取或设置导弹的当前位置

Declaration

```
public Vector3D Position { readonly get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示空间位置 坐标系：右手坐标系，X向右，Y向上，Z向前

## Speed

获取或设置导弹的当前速度大小

Declaration

```
public double Speed { readonly get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒 表示导弹运动的标量速度

## Type

获取或设置导弹的类型

Declaration

```
public MissileType Type { readonly get; set; }
```

Property Value

Type	Description
MissileType	

Remarks

用于区分不同种类的导弹 影响导弹的行为特征和性能参数

## Velocity

获取或设置导弹的当前速度向量

Declaration

```
public Vector3D Velocity { readonly get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

包含速度的大小和方向信息 单位：米/秒

[Show / Hide Table of Contents](#)

# Class SimulationElement

仿真元素的抽象基类，所有仿真中的实体都继承自此类

## Inheritance

↳ [object](#)

↳ [SimulationElement](#)

↳ [InfraredTracker](#)

↳ [LaserBeamRider](#)

↳ [LaserDesignator](#)

↳ [BaseMissile](#)

↳ [Tank](#)

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public abstract class SimulationElement
```

## Remarks

该类提供了仿真实体的基本功能：

- 位置和运动状态管理
- 生命周期管理（激活/停用）
- 事件发布机制
- 状态更新机制 所有具体的仿真实体（如导弹、目标等）都应继承此类并实现其抽象方法。

## Constructors

### SimulationElement(string, Vector3D, Orientation, double, ISimulationManager)

初始化仿真元素的新实例

## Declaration

```
protected SimulationElement(string id, Vector3D position, Orientation orientation, double speed, ISimulationManager simulationManager)
```

## Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	仿真元素的唯一标识符
<a href="#">Vector3D</a>	<i>position</i>	初始位置坐标
<a href="#">Orientation</a>	<i>orientation</i>	初始朝向角度
<a href="#">double</a>	<i>speed</i>	初始速度大小，单位：米/秒
<a href="#">ISimulationManager</a>	<i>simulationManager</i>	仿真管理器实例

Remarks

构造函数设置实体的初始状态：

- 根据ID、位置、朝向和速度初始化实体
- 计算初始速度向量
- 设置初始状态为未激活

Properties

Id

获取或设置仿真元素的唯一标识符

Declaration

```
public virtual string Id { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

ID在仿真系统中必须唯一，用于标识和查找实体

IsActive

获取仿真元素是否处于活动状态

Declaration

```
public virtual bool IsActive { get; protected set; }
```

Property Value

Type	Description
<a href="#">bool</a>	

Remarks

true表示实体当前处于活动状态，可以参与仿真 false表示实体已停用，不再参与仿真计算

Orientation

获取或设置仿真元素的当前朝向

Declaration

```
public virtual Orientation Orientation { get; set; }
```

Property Value

Type	Description
Orientation	

Remarks

使用欧拉角表示实体的朝向 包含偏航角、俯仰角和滚转角

Position

获取或设置仿真元素的当前位置

Declaration

```
public virtual Vector3D Position { get; set; }
```

Property Value

Type	Description
Vector3D	

Remarks

使用三维向量表示实体在空间中的位置 坐标系：右手坐标系，X向右，Y向上，Z向前

SimulationManager

获取或设置仿真管理器的引用

Declaration

```
protected ISimulationManager SimulationManager { get; set; }
```

Property Value

Type	Description
ISimulationManager	

Remarks

用于访问仿真系统的核心功能，如事件发布、实体查询等 在构造函数中初始化，整个生命周期内保持不变

Speed

获取或设置仿真元素的当前速度大小

Declaration

```
public virtual double Speed { get; set; }
```

Property Value

Type	Description
double	

Remarks

单位：米/秒 此属性仅表示速度的标量值，方向信息需要结合Velocity属性

Velocity

获取或设置仿真元素的当前速度向量

Declaration



```
public virtual Vector3D Velocity { get; set; }
```

Property Value

Type	Description
<a href="#">Vector3D</a>	

Remarks

单位：米/秒 包含速度的大小和方向信息 向量的模等于Speed属性值

## Methods

### Activate()

激活仿真元素，使其参与仿真

Declaration

```
public virtual void Activate()
```

Remarks

激活操作会：

- 将IsActive设置为true
- 发布实体激活事件
- 允许实体参与仿真计算

### Deactivate()

停用仿真元素，将其从仿真中移除

Declaration

```
public virtual void Deactivate()
```

Remarks

停用操作会：

- 将IsActive设置为false
- 发布实体停用事件
- 停止实体参与仿真计算

### GetStatus()

获取仿真元素的状态信息

Declaration

```
public virtual string GetStatus()
```

Returns

Type	Description
<a href="#">string</a>	包含实体当前状态的字符串描述

Remarks

返回的字符串包含：

- 实体类型

- 实体ID
- 当前位置
- 当前朝向
- 活动状态

### PublishEvent(SimulationEvent)

发布仿真事件到事件系统

Declaration

```
protected void PublishEvent(SimulationEvent evt)
```

Parameters

Type	Name	Description
<a href="#">SimulationEvent</a>	<i>evt</i>	要发布的事件实例

Remarks

在发布事件前会：

- 设置事件的发送者ID
- 添加时间戳
- 通过仿真管理器发布到事件系统

### Update(double)

更新仿真元素的状态

Declaration

```
public abstract void Update(double deltaTime)
```

Parameters

Type	Name	Description
<a href="#">double</a>	<i>deltaTime</i>	时间步长，单位：秒

Remarks

抽象方法，需要在派生类中实现具体的更新逻辑：

- 更新位置和朝向
- 计算新的速度和加速度
- 处理碰撞和其他物理效果
- 触发相关事件

# Class TargetDestroyedEvent

目标被摧毁事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [TargetDestroyedEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** [ThreatSource.dll](#)

## Syntax

```
public class TargetDestroyedEvent : SimulationEvent
```

## Properties

### TargetId

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Class LaserIlluminationUpdateEvent

激光照射更新事件，表示激光照射状态的更新

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserIlluminationUpdateEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserIlluminationUpdateEvent : SimulationEvent
```

## Remarks

用于实时更新激光照射的状态 在照射过程中周期性触发

## Properties

### LaserDesignatorId

获取或设置激光定位器的ID

#### Declaration

```
public string? LaserDesignatorId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

#### Remarks

标识正在照射的定位器设备

# TargetId

获取或设置被照射目标的ID

Declaration

```
public string? TargetId { get; set; }
```

Property Value

Type	Description
<a href="#">string</a>	

Remarks

标识正在被照射的目标实体

# Class RadiometerSensorData

辐射计传感器数据类，包含毫米波辐射探测的具体数据

## Inheritance

↳ [object](#)  
↳ [SensorData](#)  
↳ RadiometerSensorData

## Inherited Members

[SensorData.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Sensor](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class RadiometerSensorData : SensorData
```

## Remarks

该类封装了毫米波辐射计的测量结果：

- 辐射强度信息
- 目标探测状态 用于目标特征识别和态势评估

## Properties

### IsTargetDetected

获取或设置是否检测到目标

#### Declaration

```
public bool IsTargetDetected { get; set; }
```

#### Property Value

Type	Description
<a href="#">bool</a>	

#### Remarks

true表示探测到有效目标 false表示未探测到目标 用于目标存在性判断

RadiationIntensity

获取或设置探测到的辐射强度，单位： 瓦特/平方米

Declaration

```
public double RadiationIntensity { get; set; }
```

Property Value

Type	Description
double	

Remarks

记录目标的毫米波辐射强度 用于目标特征分析和识别判断

# Interface IGuidanceSystem

制导系统接口，定义了所有制导系统的通用功能

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

Syntax

```
public interface IGuidanceSystem
```

## Remarks

该接口提供了制导系统的基本功能规范：

- 制导状态判断
- 制导信息更新
- 制导加速度计算 是所有具体制导系统实现的基础

## Properties

### HasGuidance

获取是否有有效的制导信息

Declaration

```
bool HasGuidance { get; }
```

Property Value

Type	Description
<a href="#">bool</a>	

Remarks

true表示当前有可用的制导信息 false表示无法获得有效制导 用于判断制导系统的工作状态

## Methods

### GetGuidanceAcceleration()

获取制导加速度指令

Declaration

```
Vector3D GetGuidanceAcceleration()
```



Returns

Type	Description
Vector3D	三维制导加速度向量，单位：米/平方秒

Remarks

返回数据：

- X分量：横向制导加速度
- Y分量：垂直制导加速度
- Z分量：纵向制导加速度 用于导弹的轨迹控制

Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

Declaration

```
void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

Parameters

Type	Name	Description
double	<i>deltaTime</i>	自上次更新以来的时间间隔，单位：秒
Vector3D	<i>missilePosition</i>	导弹当前位置，单位：米
Vector3D	<i>missileVelocity</i>	导弹当前速度，单位：米/秒

Remarks

更新过程：

- 获取最新目标信息
- 计算制导参数
- 更新制导状态
- 生成制导指令

# Class TestSimulationAdapter

用于测试的仿真环境适配器

## Inheritance

↳ [object](#)

↳ TestSimulationAdapter

## Implements

[ISimulationAdapter](#)

## Inherited Members

[object.Equals\(object\)](#)

[object.Equals\(object, object\)](#)

[object.GetHashCode\(\)](#)

[object.GetType\(\)](#)

[object.MemberwiseClone\(\)](#)

[object.ReferenceEquals\(object, object\)](#)

[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation.Testing](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class TestSimulationAdapter : ISimulationAdapter
```

## Constructors

### TestSimulationAdapter(ISimulationManager)

#### Declaration

```
public TestSimulationAdapter(ISimulationManager simulationManager)
```

#### Parameters

Type	Name	Description
<a href="#">ISimulationManager</a>	<i>simulationManager</i>	

## Methods

### AddTestEntity(string, object)

#### Declaration

```
public void AddTestEntity(string id, object entity)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	
<a href="#">object</a>	<i>entity</i>	

## ClearEvents()

Declaration

```
public void ClearEvents()
```

## ClearTestEntities()

Declaration

```
public void ClearTestEntities()
```

## GetEntity(string)

从外部仿真环境获取实体

Declaration

```
public object? GetEntity(string id)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	实体ID

Returns

Type	Description
<a href="#">object</a>	实体对象，如果不存在则返回null

## GetPublishedEvents()

Declaration

```
public IReadOnlyList<object> GetPublishedEvents()
```

Returns

Type	Description
<a href="#">IReadOnlyList&lt;object&gt;</a>	

## GetReceivedEvents()

Declaration

```
public IReadOnlyList<object> GetReceivedEvents()
```

Returns

Type	Description
<a href="#">IReadOnlyList&lt;object&gt;</a>	

**PublishToExternalSimulation<T>(T)**

发布事件到第三方仿真环境

Declaration

```
public void PublishToExternalSimulation<T>(T evt)
```

Parameters

Type	Name	Description
T	<i>evt</i>	要发布的事件

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

用于将威胁源仿真库中的事件同步到外部仿真环境

**ReceiveFromExternalSimulation<T>(T)**

从第三方仿真环境接收事件

Declaration

```
public void ReceiveFromExternalSimulation<T>(T evt)
```

Parameters

Type	Name	Description
T	<i>evt</i>	接收到的事件

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

用于接收外部仿真环境的事件并在威胁源仿真库中处理

Implements

[ISimulationAdapter](#)

# Class MillimeterWaveGuidanceSystem

毫米波导引头系统类，实现了基于毫米波雷达的目标探测和跟踪功能

## Inheritance

↳ [object](#)  
↳ [BasicGuidanceSystem](#)  
↳ [MillimeterWaveGuidanceSystem](#)

## Implements

[IGuidanceSystem](#)

## Inherited Members

[BasicGuidanceSystem.HasGuidance](#)  
[BasicGuidanceSystem.MaxAcceleration](#)  
[BasicGuidanceSystem.ProportionalNavigationCoefficient](#)  
[BasicGuidanceSystem.Position](#)  
[BasicGuidanceSystem.Velocity](#)  
[BasicGuidanceSystem.GuidanceAcceleration](#)  
[BasicGuidanceSystem.GetGuidanceAcceleration\(\)](#)  
[BasicGuidanceSystem.CalculateGuidanceAcceleration\(double\)](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Guidance](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveGuidanceSystem : BasicGuidanceSystem, IGuidanceSystem
```

## Remarks

该类提供了毫米波导引头系统的核心功能：

- 目标探测和识别
- 信噪比计算
- 抗干扰处理
- 比例导引控制 用于实现全天候制导打击

## Constructors

### MillimeterWaveGuidanceSystem(double, double, ISimulationManager)

初始化毫米波导引头系统的新实例

## Declaration

```
public MillimeterWaveGuidanceSystem(double maxAcceleration, double guidanceCoefficient, I
SimulationManager simulationManager)
```

Parameters

Type	Name	Description
double	maxAcceleration	最大加速度，单位：米/平方秒
double	guidanceCoefficient	制导系数
ISimulationManager	simulationManager	仿真管理器实例

Remarks

- 构造过程：
- 初始化基类参数
  - 设置仿真管理器
  - 初始化目标位置
  - 注册干扰事件处理

## Properties

### SimulationManager

获取或设置仿真管理器实例

Declaration

```
public ISimulationManager SimulationManager { get; set; }
```

Property Value

Type	Description
ISimulationManager	

Remarks

用于获取场景中的目标信息 实现目标探测功能

## Methods

### GetStatus()

获取制导系统的详细状态信息

Declaration

```
public override string GetStatus()
```

Returns

Type	Description
string	包含完整状态参数的字符串

Overrides

[BasicGuidanceSystem.GetStatus\(\)](#)

Remarks

- 返回信息：
- 基本状态信息

- 目标位置 用于系统监控和调试

## Update(double, Vector3D, Vector3D)

更新制导系统的状态和计算结果

### Declaration

```
public override void Update(double deltaTime, Vector3D missilePosition, Vector3D missileVelocity)
```

### Parameters

Type	Name	Description
double	deltaTime	自上次更新以来的时间间隔，单位：秒
Vector3D	missilePosition	导弹当前位置，单位：米
Vector3D	missileVelocity	导弹当前速度，单位：米/秒

### Overrides

[BasicGuidanceSystem.Update\(double, Vector3D, Vector3D\)](#)

### Remarks

更新过程：

- 探测目标位置
- 计算目标速度
- 生成制导指令
- 限制最大加速度

## Implements

[IGuidanceSystem](#)

# Class LaserBeamStopEvent

激光波束停止事件，表示激光波束制导结束

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [LaserBeamStopEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class LaserBeamStopEvent : SimulationEvent
```

## Remarks

用于通知系统激光波束制导已结束 触发时机：停止发射制导激光波束时

# Properties

## LaserBeamRiderId

获取或设置激光波束制导器的ID

## Declaration

```
public string? LaserBeamRiderId { get; set; }
```

## Property Value

Type	Description
<a href="#">string</a>	





[Show / Hide Table of Contents](#)

# Namespace ThreatSource.Simulation.Testing

## Classes

### **TestSimulationAdapter**

用于测试的仿真环境适配器

[Show / Hide Table of Contents](#)

# Class UltravioletWarnerAlarmEvent

紫外告警器警报事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ UltravioletWarnerAlarmEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class UltravioletWarnerAlarmEvent : SimulationEvent
```

## Properties

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

[Show / Hide Table of Contents](#)

# Class InfraredGuidanceMissileLightEvent

红外指令制导导弹点亮红外热源事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ InfraredGuidanceMissileLightEvent

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class InfraredGuidanceMissileLightEvent : SimulationEvent
```

## Remarks

用于模拟导弹尾部红外热源的激活 触发时机：导弹发动机点火时

## Properties

### RadiationPower

获取或设置红外热源辐射功率

#### Declaration

```
public double RadiationPower { get; set; }
```

#### Property Value

Type	Description
<a href="#">double</a>	

#### Remarks

单位：瓦特 表示导弹尾焰的红外辐射强度

# Class SimulationManager

仿真管理器的默认实现，提供仿真系统的核心功能

## Inheritance

↳ [object](#)  
↳ [SimulationManager](#)

## Implements

[ISimulationManager](#)

## Inherited Members

[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class SimulationManager : ISimulationManager
```

## Remarks

该类实现了ISimulationManager接口，提供：

- 事件系统：支持发布/订阅模式的事件处理
- 实体管理：实体的注册、注销和查询
- 第三方集成：支持与外部仿真环境的对接

线程安全说明：

- 实体管理相关操作使用锁机制确保线程安全
- 事件系统的操作不保证线程安全，应在单线程环境中使用

## Methods

### GetAllEntities()

获取仿真系统中的所有实体

#### Declaration

```
public IReadOnlyList<object> GetAllEntities()
```

#### Returns

Type	Description
<code>ReadOnlyList&lt;object&gt;</code>	所有实体的列表

Remarks

只返回本地注册的实体，不包括外部系统的实体 返回的列表是原始集合的副本，可以安全地遍历

线程安全：使用锁保护查询操作

GetEntitiesByType<T>()

获取特定类型的所有实体

Declaration

```
public IReadOnlyList<T> GetEntitiesByType<T>() where T : class
```

Returns

Type	Description
<code>ReadOnlyList&lt;T&gt;</code>	指定类型的实体列表

Type Parameters

Name	Description
<i>T</i>	实体类型

Remarks

只返回本地注册的实体，不包括外部系统的实体 返回的列表是原始集合的副本，可以安全地遍历

线程安全：使用锁保护查询操作

GetEntityById(string)

根据ID获取实体

Declaration

```
public object? GetEntityById(string id)
```

Parameters

Type	Name	Description
<code>string</code>	<i>id</i>	实体ID

Returns

Type	Description
<code>object</code>	实体对象，如果不存在则返回null

Remarks

查找顺序：

- 1. 在本地实体集合中查找
- 2. 如果未找到且配置了外部适配器，尝试从外部系统获取

线程安全：使用锁保护查询操作

GetSimulationAdapter()

获取当前的仿真环境适配器

Declaration

```
public ISimulationAdapter? GetSimulationAdapter()
```

Returns

Type	Description
<a href="#">ISimulationAdapter</a>	当前配置的适配器实例，如果未配置则返回null

## PublishEvent<T>(T)

发布事件到仿真系统

Declaration

```
public void PublishEvent<T>(T evt)
```

Parameters

Type	Name	Description
T	<i>evt</i>	要发布的事件实例

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

发布过程：

1. 查找并调用所有注册的事件处理器
2. 如果配置了外部仿真适配器，同时发布到外部系统

## RegisterEntity(string, object)

注册实体到仿真系统

Declaration

```
public bool RegisterEntity(string id, object entity)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	实体ID
<a href="#">object</a>	<i>entity</i>	实体对象

Returns

Type	Description
<a href="#">bool</a>	注册是否成功

Remarks

注册失败的情况：

- ID为空或实体为null
- ID已被其他实体使用

线程安全：使用锁保护注册操作

## SetSimulationAdapter(ISimulationAdapter)

设置第三方仿真环境适配器

Declaration

```
public void SetSimulationAdapter(ISimulationAdapter adapter)
```

Parameters

Type	Name	Description
<a href="#">ISimulationAdapter</a>	<i>adapter</i>	要设置的适配器实例

Remarks

设置适配器后，仿真系统将：

- 在发布事件时同步到外部系统
- 在查询实体时同时查询外部系统

Exceptions

Type	Condition
<a href="#">ArgumentNullException</a>	适配器参数为null时抛出

SubscribeToEvent<T>(Action<T>)

订阅特定类型的事件

Declaration

```
public void SubscribeToEvent<T>(Action<T> handler)
```

Parameters

Type	Name	Description
<a href="#">Action&lt;T&gt;</a>	<i>handler</i>	事件处理函数

Type Parameters

Name	Description
<i>T</i>	事件类型

Remarks

如果是首次订阅该类型事件，会创建新的处理器列表 同一个处理器可以多次订阅，会被多次调用

UnregisterEntity(string)

从仿真系统注销实体

Declaration

```
public bool UnregisterEntity(string id)
```

Parameters

Type	Name	Description
<a href="#">string</a>	<i>id</i>	要注销的实体ID

Returns

Type	Description
<a href="#">bool</a>	注销是否成功

Remarks

如果实体不存在，返回false 线程安全：使用锁保护注销操作

# UnsubscribeFromEvent<T>(Action<T>)

取消订阅特定类型的事件

## Declaration

```
public void UnsubscribeFromEvent<T>(Action<T> handler)
```

## Parameters

Type	Name	Description
Action<T>	handler	要取消的事件处理函数

## Type Parameters

Name	Description
T	事件类型

## Remarks

如果取消订阅后该类型没有其他处理器，会删除整个处理器列表 如果处理器不存在，操作会被忽略

## Implements

[ISimulationManager](#)



[Show / Hide Table of Contents](#)

# Class MillimeterWaveWarnerAlarmStopEvent

毫米波告警器警报停止事件

## Inheritance

↳ [object](#)  
↳ [SimulationEvent](#)  
↳ [MillimeterWaveWarnerAlarmStopEvent](#)

## Inherited Members

[SimulationEvent.SenderId](#)  
[SimulationEvent.Timestamp](#)  
[object.Equals\(object\)](#)  
[object.Equals\(object, object\)](#)  
[object.GetHashCode\(\)](#)  
[object.GetType\(\)](#)  
[object.MemberwiseClone\(\)](#)  
[object.ReferenceEquals\(object, object\)](#)  
[object.ToString\(\)](#)

**Namespace:** [ThreatSource.Simulation](#)

**Assembly:** ThreatSource.dll

## Syntax

```
public class MillimeterWaveWarnerAlarmStopEvent : SimulationEvent
```

## Properties

### TargetId

目标ID

#### Declaration

```
public string? TargetId { get; set; }
```

#### Property Value

Type	Description
<a href="#">string</a>	

# Namespace ThreatSource.Utils

## Classes

### **MotionAlgorithm**

运动算法静态类，提供各种运动计算方法

### **Vector3D**

表示三维空间中的向量，提供基本的向量运算功能

## Structs

### **Orientation**

表示三维空间中的方向，使用欧拉角表示

### **Vector2D**

表示二维平面上的向量

[Show / Hide Table of Contents](#)

# Enum MissileType

导弹类型枚举，定义了系统支持的所有导弹类型

**Namespace:** [ThreatSource.Missile](#)

**Assembly:** ThreatSource.dll

Syntax

```
public enum MissileType
```

## Remarks

包含以下导弹类型：

- 标准导弹：基础型号导弹
- 激光半主动制导：利用目标反射的激光能量进行制导
- 激光驾束制导：沿着激光束路径飞行
- 红外指令制导：通过红外信号接收指令进行制导
- 红外成像末制导：末段使用红外成像进行制导
- 毫米波末制导：末段使用毫米波雷达进行制导
- 末敏弹：末段敏感引信的导弹

## Fields

Name	Description
InfraredCommandGuidance	
InfraredImagingTerminalGuidance	
LaserBeamRiderGuidance	
LaserSemiActiveGuidance	
MillimeterWaveTerminalGuidance	
StandardMissile	
TerminalSensitiveMissile	