ArcGIS Pro SDK for .NET: An Overview of the Utility Network Management API

Rich Ruh
David Crawford
Today’s Agenda

• Getting Started
• A Survey of the Utility Network APIs
• Things to Look Out for in Pro 2.1
• Plans for Pro 2.2
• Learning More
Getting Started
https://pro.arcgis.com/en/pro-app/sdk/

Search for: “ArcGIS Pro SDK”
Overview

- The utility network C# SDK is a managed .NET SDK that provides access to the utility network.
- It is an object-oriented SDK that aligns with modern C# practices and existing frameworks.
- It adheres to the principles and architecture of the general Pro SDK.
- This presentation assumes a basic understanding of the utility network information model.
Architectural Topics

- DML-only (Data Manipulation Language)
- Threading
The utility network API is a **DML-only (Data Manipulation Language)** API

- Schema creation and modification operations such as creating domain networks, adding and deleting rules, etc., need to performed using Python
- This is in alignment with the rest of the Geodatabase API
- Python can be called from C# by using the Geoprocessing API

```csharp
var args = Geoprocessing.MakeValueArray(utilityNetworkPath, @"ALL", @"rules.csv");
var result = Geoprocessing.ExecuteToolAsync("un.ImportRules", args);
```
Architecture: Threading

- Almost all of the methods in the utility network API should be called on the MCT (Main CIM Thread)

```csharp
Task t = QueuedTask.Run(() =>
{
   //put utility network code here
});
```
Other Ways to Access the Utility Network

• In addition to the ArcGIS Pro Managed SDK, there are other ways to program against a utility network:
  - Geoprocessing models and Python scripts
  - Directly coding against the REST APIs

```python
# Update subnetworks
arcpy.AddMessage("Update subnetworks")
arcpy.UpdateSubnetwork_un(utilityNetwork, domainNetworkName, "Subtransmission", "ALL_SUBNETWORKS_IN_TIER")
arcpy.UpdateSubnetwork_un(utilityNetwork, domainNetworkName, "Medium Voltage", "ALL_SUBNETWORKS_IN_TIER")
arcpy.UpdateSubnetwork_un(utilityNetwork, domainNetworkName, "Low Voltage Mean", "ALL_SUBNETWORKS_IN_TIER")
arcpy.AddMessage("Finished updating subnetworks")
```
A Survey of the Utility Network APIs
Organization of the Utility Network API

- The API can be logically divided into nine different sections.
- The diagram at right provides a functional organization of the API:
  - Strictly speaking, the API is a collection of classes.
  - Not a layered architecture.
Organization of the Utility Network API

• The Utility Network Class is the root object that provides access to the utility network API
The UtilityNetwork Class

- Serves as the central hub of the utility network API
- Can be obtained from
  - A geodatabase
    - Geodatabase.OpenDataset<UtilityNetwork>(string datasetName) : UtilityNetwork
  - A feature class or table
    - Table.GetControllerDataset() : IReadOnlyList<Dataset>
  - A utility network layer
    - GetUtilityNetwork() : UtilityNetwork
Obtaining a UtilityNetwork object

Demo
Organization of the Utility Network API

- Definition and Schema describes the classes and methods that provide information about the utility network schema
These classes provide read-only access to schema information
These classes are value objects that are derived from information cached with the feature service
Encryption of the Utility Network API

- Element covers the basic encapsulation of a row in the utility network API.
Elements

- **The Element class** represents a row inside a utility network, *plus* a terminal (if applicable).

- **Used throughout the API:**
  - Elements are used to create and delete associations.
  - Elements specify starting points and barriers for use with tracing.
  - Elements are returned as results from traces.
  - Etc.

- **Created using `CreateElement()` factory methods on the `UtilityNetwork` class.**
Organization of the Utility Network API

- Network Topology covers routines that query the topological index
Utility Network Topology

- The network topology stores connectivity, containment, and attachment information used by the utility network to facilitate fast network traversal/analytical operations.
- Network topology is constructed from:
  - Geometric coincidence and...
  - Associations in combination with...
  - A powerful rules engine.
- Topology is updated and validated with the `ValidateNetworkTopology()` method on the `UtilityNetwork` class.
- Access to fine-grained topology is not provided.
Organization of the Utility Network API

- Associations covers routines that query and edit associations between utility network rows
Associations

• Association Types
  - Connectivity
  - Containment
  - Structural Attachment

• Query
  - Get associations for a given Element

• Edit
  - Create or delete associations
Organization of the Utility Network API

- Subnetworks provides classes and routines to query and edit utility network subnetworks
Subnetworks

- The management of subnetworks allows organizations to optimize the delivery of resources and track the status of a network.
- A single subnetwork can be used to model such things as a circuit in electric networks, and a zone in gas and water networks.
- The `SubnetworkManager` is a class that contains a collection of subnetwork management routines.
Subnetwork Manager

• Query
  - GetSubnetworks() allows retrieval of Subnetworks based on their state
• Edit
  - Add and remove controllers with EnableController() and DisableController()
The Subnetwork Class

- Represents a subnetwork
- Update a subnetwork
- Fetch the SubnetLine feature
- Return set of system network diagrams associated with this subnetwork
Organization of the Utility Network API

- Tracing provides tracing functionality
- Tracing entails assembling a subset of utility network elements that meet a specified criteria
- Tracing uses network data to provide business value to utilities
  - Answers questions and solves problems about the current state of the network
  - Helps design future facilities
  - Helps organize business practices
Components of a Trace

- Different kinds of traces are implemented with Tracer objects
Components of a Trace

- A Trace Argument encapsulates the input parameters for a trace

Trace Argument
- Starting Points
- Barriers
- Result Types
- Trace Configuration

Tracer
Components of a Trace

- The Tracer generates a set of Result objects as output

Tracer arguments:
- Trace Configuration
- Result Types
- Barriers
- Starting Points

Result objects
Components of a Trace

We’ll cover each of these parts in more detail.

1. Tracer
2. Trace Argument
   - Starting Points
   - Barriers
   - Result Types
   - Trace Configuration
3. Result objects
• Tracers define the tracing algorithm to be used
• Tracer objects are created using TraceManager
Tracer Objects

- **Tracer**
  - Abstract Class
  - Properties
    - Name (get): string
    - UtilityNetwork (get): UtilityNetwork
  - Methods
    - Trace(TraceArgument traceArgument): TraceResult

- **ConnectedTracer**
  - Class: Tracer

- **SubnetworkTracer**
  - Class: Tracer

- **SubnetworkControllerTracer**
  - Class: Tracer

- **UpstreamTracer**
  - Class: Tracer

- **DownstreamTracer**
  - Class: Tracer

- **LoopsTracer**
  - Class: Tracer

- **ShortestPathTracer**
  - Class: Tracer
The `TraceArgument` class consolidates trace parameters:
- Starting Points
- Barriers
- Result Types
- Trace Configuration
Trace Configuration – Basic Properties

IncludeContainers: bool
IncludeContent: bool
IncludeStructures: bool
IncludeBarriersWithResults: bool
DomainNetwork: DomainNetwork
SourceTier: Tier
TargetTier: Tier
ValidateConsistency: bool
Trace Configuration – Traversability

- Traversability is based on
  - Barriers
    - Comparisons of network attributes (NetworkAttributeComparison), or
    - Checking for the existence of a Category (CategoryComparison)
  - These can be combined with boolean And and Or operations to form more complex filters
  - Function Barriers
    - Evaluation of a functional expression
Trace Configuration – Functions

- The caller can specify a collection of functions for a trace
  - These functions calculate values based on a network attribute
- At the conclusion of the trace, function results can be obtained
Trace Configuration – Filters

- Filters are a mechanism to stop tracing when returning results
- They do not stop traversability to the controller
- For example, returning the next upstream protective device
  - Create a filter as follows:

- If you tried to do this with traversability, the trace would fail because this condition would prevent finding the subnetwork source
Filters

- There are a number of different ways to define filters
  - Use a Condition based on network attributes or categories
  - Use a functional expression
  - Use a bitset filter
  - Use a nearest neighbor filter
Trace Configuration – Output

- Output filtering is applied as the final step of the tracing process
  - It takes place after traversability, filtering, and function calculation
- Two kinds of output filtering are provided
  - Output Condition
  - Output Asset Types
Trace Configuration – Propagators

- A propagator defines the propagation of a network attribute along a traversal, as well as provide a filter to stop traversal.
- Propagators are only applicable to subnetwork-based traces (subnetwork, subnetwork source, upstream, downstream).
- The canonical example is phase propagation - open devices along the network will restrict some phases from continuing along the trace.
How Propagators Work

- Propagator values are computed as a pre-process step before the main trace takes place
  - Starting at each controller, the propagator uses its PropagatorFunction and NetworkAttribute to calculate a value at each element
  - This pre-process traversal covers the extent of a subnetwork
- During the trace itself, propagator filters are tested at the same time as traversal filters
Information is returned from a trace operation with a set of Result objects.
Trace Results

- The Trace() method on the Tracer class returns a set of Result objects.
- One Result object is returned for each ResultType specified in TraceArgument.ResultTypes.
Returning a List of Element Objects

- The `ElementResult` class contains a list of `Element` objects

Elements : `IReadOnlyList<Element>`
Returning Function Results

- The `FunctionOutputResult` class contains a list of `FunctionOutput` objects
- The `FunctionOutput` class is a set of Function-Value pairs
  - Function : Function
  - Value : object
Future: Additional Result Types

- Future software versions may implement additional result types.
- Some possible result types that might be supported in the future are:
  - Propagator values per row
  - Connectivity information
  - Network Diagram
  - Geometry
  - Additional network attributes
- Each result type will add a value to the `ResultType` enum and a new concrete subclass of `Result`
Organization of the Utility Network API

- Network Diagrams allows the developer to query and edit network diagrams
The DiagramManager Class

- The DiagramManager class serves as the core class in the network diagrams API.
- Use the DiagramManager class to:
  - Create network diagrams
  - Retrieve diagram templates
  - Retrieve network diagrams
  - Retrieve the related network
The NetworkDiagram Class

- This class represents a diagram generated from a portion of the utility network
- Some key routines:
  - Update
  - Append
  - Overwrite
  - Store
  - Delete
Diagram Element Classes

- A network diagram consists of a set of diagram elements which are either junctions, edges, or containers.
Retrieving Diagram Elements

- By Type (edge, junction, container)
  - DiagramElementQueryByElementTypes class
- By Extent
  - DiagramElementQueryByExtent class
- By set of ObjectIDs
  - DiagramElementQueryByObjectIDs class
Custom Layouts

1. Query for set of diagram elements
2. Change the shape of those elements
3. Add the diagram elements to a `NetworkDiagramSubset` object
4. Pass that as an argument to `NetworkDiagram.SaveLayout()`
Network Diagrams
Demo
Finally, Pro Integration describes how the utility network API integrates with other parts of the Pro SDK.
The UtilityNetworkLayer Class

- The **UtilityNetworkLayer** class represents a utility network layer in a map.
- The **UtilityNetworkLayer** class inherits from **CompositeLayer**, which can be used to iterate through the dirty area and error sublayers.

**GetUtilityNetwork() : UtilityNetwork**

- Returns the **UtilityNetwork** class pointed at by this layer.
- Used with ArcGIS Pro add-ins to obtain the underlying geodatabase object from the selected layer.
Ribbon Integration

- When you are writing a button or tool to appear on the ribbon, the config.daml file is used to configure when the object is enabled
  - As described by the Pro SDK documentation

- We provide a utility network condition to enable it if any of the following layers are selected:
  - Utility Network Layer
  - Feature Layer that corresponds to a feature class that belongs to a utility network
  - Subtype Group Layer that corresponds to a feature class that belongs to a utility network

condition="esri_mapping_utilityNetworkLayerSelectedCondition"
Editor Integration

• The preferred technique for editing is to use the high level `EditOperation` class
  - This class includes a number of high-level editing routines like `Create()`, `Delete()`, `Move()`
  - These routines queue up edits
  - `EditOperation.Execute()` fires off the actual edits

• The `EditOperation` was extended in 2.1 to support utility network edits

Read the [Editor Conceptual Documentation](#) for more information
Creating and Deleting Associations

• New set of AssociationDescription classes created
  - ConnectivityAssociationDescription
  - ContainmentAssociationDescription
  - StructuralAttachmentAssociationDescription

• New Create() and Delete() overloads created on EditOperation class
Creating Features and Associations in the Same Edit Operation

• How do you specify an association between two features when the features haven’t been created yet?
  - You don’t have an ObjectID or GlobalID
• New CreateEx() methods on EditOperation return a RowToken
  - This token can be passed into AssociationDescription classes
Editing with low-level Geodatabase Routines

- We also provide low-level geodatabase objects for editing
- This is the only way to edit in a stand-alone application
- Not recommended in an ArcGIS Pro Add-in
  - The Map is not refreshed
  - The Undo/Redo stack is not modified
Editing
Demo
Things to Look Out For in Pro 2.1
Things to Look Out For in Pro 2.1

- Some utility network operations are not well integrated into Pro
  - Core.Data vs. Core.Desktop
- Some functionality is missing
Pro Integration Shortcomings

• ValidateNetworkTopology
  - Cannot be in an edit session
  - Doesn’t integrate with Pro undo/redo stack
  - No screen redraw
  - Workaround: Use Geoprocessing

• EnableController/DisableController
  - Doesn’t integrate with Pro undo/redo stack
  - No screen redraw
  - Workaround: None

• Update Subnetwork
  - No screen redraw
  - Workaround: Use Geoprocessing
Missing Functionality

• Definition and Schema
  - Tier groups
  - Terminal paths
  - Attribute rules

• Associations
  - View Association tool

• Subnetworks
  - Update All
  - Export
  - General Queries
Plans for Pro 2.2
Plans for Pro 2.2

- Some utility network operations are not well integrated into Pro
  - Core.Data vs. Core.Desktop
- Some functionalities missing

- Better integrate utility network operations with Pro
- Add some missing functionality
- Support other new utility network functionality
Fixing Pro Integration Shortcomings

• Add utility network extension methods
  - Add a using statement for ArcGIS.Core.Data.UtilityNetwork.Extensions to add C# extension methods
  - UtilityNetwork class
    - ValidateNetworkTopologyInEditOperation()
  - SubnetworkManager class
    - EnableControllerInEditOperation()
    - DisableControllerInEditOperation()

• Add method to redraw screen
  - MapView.Redraw()
Add Missing Functionality

- Tier groups
- Terminal paths
- Attribute rules
- Viewing associations
- General subnetwork queries

These items are a work-in-progress. They are ordered according to current priorities.

No warranties for their inclusion in 2.2 are expressed or implied.
Learning More
Learn key concepts

Framework
Build add-ins which extend the Pro UI and leverage asynchronous programming.

Editing
Create custom tools to construct and edit feature classes with your unique logic.

Map exploration
Create capabilities to navigate and explore the map in 2D and 3D.

Content
Manage Pro project content items such as maps, layouts, styles, and more.

Utility Network
Develop custom network traces, editing tools and diagrams.

https://pro.arcgis.com/en/pro-app/sdk/
The utility network is a comprehensive framework of functionality in ArcGIS for modeling utility systems such as electric, gas, water, storm water, wastewater, and telecommunications. This topic provides an introduction to the utility network API. It details the classes and methods that query and edit the utility network. The utility network API is commonly used in conjunction with the geodatabase and editing.

**Language:**  C# and Visual Basic  
**Subject:** Utility Network  
**Contributor:** ArcGIS Pro SDK Team <arcgisprosdk@esri.com>  
**Organization:** Esri, http://www.esri.com  
**Date:** 01/15/2018  
**ArcGIS Pro:** 2.1  
**Visual Studio:** 2015, 2017

In this topic
- Introduction
  - Overview
The utility network is a comprehensive framework of functionality in ArcGIS for modeling utility systems such as electric, gas, water, storm water, wastewater, and telecommunications. This topic provides an introduction to the utility network API. It details the classes and methods that query and edit the utility network. The utility network API is commonly used in conjunction with the geodatabase and editing.
Samples

• Categories Usage
  - Demonstrates using the schema classes to list all of the asset types that support a given category

• Load Report Sample
  - Does a downstream trace from a starting point and creates a report that sums the load and counts the customers per phase

• Place a Transformer Bank
  - Targeted for Spring 2018

• Custom Network Diagram Layout
  - Targeted for Spring 2018

• Suggestions?
• Write your own?
The utility network is a comprehensive framework of functionality in ArcGIS for modeling utility systems such as electric, gas, water, storm water, wastewater, and telecommunications. This topic provides an introduction to the utility network API. It details the classes and methods that query and edit the utility network. The utility network API is commonly used in conjunction with the geodatabase and editing.

- **Language:** C# and Visual Basic
- **Subject:** Utility Network
- **Contributor:** ArcGIS Pro SDK Team <arcgisprosdk@esri.com>
- **Organization:** Esri, http://www.esri.com
- **Date:** 01/15/2018
- **ArcGIS Pro:** 2.1
- **Visual Studio:** 2015, 2017

**In this topic**
- Introduction
  - Overview

**Developing with ArcGIS Pro**
- Requirements
- Installing ArcGIS Pro SDK for .NET
- Getting started
- ProConcepts: Migrating to ArcGIS Pro
- ProSnippets
- ArcGIS Pro API
- Release notes
Snippets

• Small pieces of code that show how to accomplish a specific task

UtilityNetwork Snippets

• Get a Utility Network from a Table
• Get a Utility Network from a Layer
• Find a Tier given a Domain Network Name and Tier Name
• Update all dirty subnetworks in a tier
• Creating a DownstreamTracer
• Create a Trace Argument
• Create a Condition to compare a Network Attribute against a set of values
• Create a Function
• Create a FunctionBarrier
• Creating an output condition
• Creating a Propagator
• Using Function Results
• Get a list of Inconsistent Network Diagrams
• Retrieving Diagram Elements
• Changing the Layout of a Network Diagram

Get a Utility Network from a Layer

```csharp
// This routine obtains a utility network from a FeatureLayer, SubtypeGroupLayer,
public static UtilityNetwork GetUtilityNetworkFromLayer(Layer layer)
{
    if (layer is UtilityNetworkLayer)
    {
        UtilityNetworkLayer utilityNetworkLayer = layer as UtilityNetworkLayer;
        return utilityNetworkLayer.GetUtilityNetwork();
    }
    else if (layer is SubtypeGroupLayer)
    {
        CompositeLayer compositeLayer = layer as CompositeLayer;
        return GetUtilityNetworkFromLayer(compositeLayer.Layers.First());
    }
    else if (layer is FeatureLayer)
    {
        FeatureLayer featureLayer = layer as FeatureLayer;
        using (FeatureClass featureClass = featureLayer.GetFeatureClass())
        {
            if (featureClass.IsControllerDatasetSupported())
            {
                IReadonlyList<Dataset> controllerDatasets = featureClass.GetControllerDataset();
                foreach (Dataset controllerDataset in controllerDatasets)
                {
                    if (controllerDataset is UtilityNetwork)
                    {
                        return controllerDataset as UtilityNetwork;
                    }
                }
            }
        }
    }
    return null;
}
```
ArcGIS Pro SDK for Microsoft .NET

Extend ArcGIS Pro using the ArcGIS Pro SDK for Microsoft .NET. Develop add-ins and solution configurations to create a custom Pro UI and user experience for your organization. Download within Visual Studio or at My Esri.
This namespace contains types used by the utility network.

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssetGroup</td>
<td>The AssetGroup class provides information about Asset Groups within the utility network. In the core geodatabase, they are implemented as subtypes.</td>
</tr>
<tr>
<td>AssetType</td>
<td>Gets information about the definition of an Asset Type.</td>
</tr>
<tr>
<td>Association</td>
<td>Represents a connectivity, containment, or structural attachment association.</td>
</tr>
<tr>
<td>DomainNetwork</td>
<td>The DomainNetwork class is used to represent a domain network inside a utility network. A domain network typically represents an industry domain such as 'Electric Distribution', 'Gas', or 'Water.' DomainNetwork objects can be obtained by calling UtilityNetworkDefinition.GetDomainNetworks().</td>
</tr>
<tr>
<td>Element</td>
<td>Represents a row inside a utility network.</td>
</tr>
<tr>
<td>NetworkAttribute</td>
<td>The NetworkAttribute class is used to represent a network attribute inside a utility network. Network attributes correspond to weights in the geometric network. NetworkAttribute objects can be obtained by calling UtilityNetworkDefinition.GetNetworkAttributes or UtilityNetworkDefinition.GetNetworkAttribute.</td>
</tr>
<tr>
<td>NetworkAttributeAssignment</td>
<td>Describes an assignment of a NetworkAttribute to a particular ArcGIS.Core.Data.Field of a NetworkSource.</td>
</tr>
<tr>
<td>NetworkSource</td>
<td>Represents a network source in a utility network.</td>
</tr>
<tr>
<td>Rule</td>
<td>Represents a rule in the utility network. These define how features can be associated with each other through connectivity, containment, and attachment.</td>
</tr>
<tr>
<td>RuleElement</td>
<td>Represents an element of a utility network rule. Each element represents one participant in an association.</td>
</tr>
</tbody>
</table>
The `UtilityNetwork` class provides an abstraction of this controller dataset. Methods on this class provide an entry point to the other areas of the utility network API.

As with other datasets in the geodatabase, a `UtilityNetwork` object can be obtained by calling `Geodatabase.OpenDataset()` or `UtilityNetwork` objects can also be obtained from a table or feature class that belongs to a utility network by using `Table.GetControllerDatasets()`.

A `UtilityNetwork` object can be obtained from a table as follows:

```csharp
public static UtilityNetwork GetUtilityNetworkFromTable(Table table)
{
    if (table.IsControllerDatasetSupported())
    {
        IReadonlyList<ControllerDataset> controllerDatasets = table.GetControllerDatasets();
        foreach (Dataset controllerDataset in controllerDatasets)
        {
            if (controllerDataset is UtilityNetwork)
            {
                return controllerDataset as UtilityNetwork;
            }
        }
    }
}
```
Questions?
rruh@esri.com
dcrawford@esri.com