ArcGIS Pro SDK for .NET:
Plugin Datasources, Deep Dive

Wolf Kaiser

2021 ESRI
DEVELOPER SUMMIT

Session Overview

- Plugin Datasources: Introduction and Architecture
- Sample of a Plugin Datasource
- Implementing a Plugin Datasource
  - Plugin Datasource template classes
- Supporting Queries (where clause) in a Plugin Datasource
- Combining Plugin Datasources with ‘Custom Project Item’ functionality

Plugin Datasources: Introduction

- ArcGIS Pro supports spatial and tabular data from many data sources and formats:
  - File based data such as Shape File, File Geodatabase, etc.
  - Relational databases like Oracle, SQL Server, IBM DB2, etc.
  - Web based services like ArcGIS Server, OGC Web services
- But many other formats are not supported
- The “ArcGIS Pro Plugin DataSource” extensibility pattern is used to integrate those unsupported formats like:
  - Unsupported relational databases such as MySQL
  - Non-relational databases such as MongoDB
  - Many other proprietary or obscure file-based data stores or web services

Plugin Datasources: Introduction

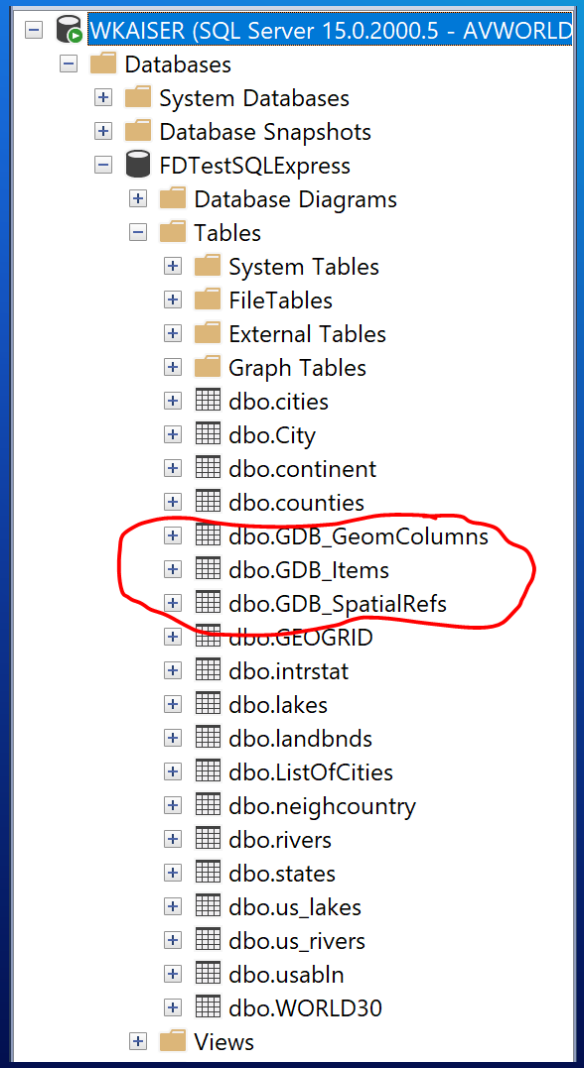
- With the Plugin Datasource framework you can:
  - Show custom data as a feature class or as a standalone table
    - Features and rows are supported
  - Consume any format and data source (file, database, web)
- Plugin Datasource limitations
  - Access in Pro is read-only
  - Access is in form of tables or feature classes
  - Access is always defined by a file or folder so either:
    - My custom data source is a file or a folder
    - My ‘connection definition’ to my custom datasource is in a file (i.e. .sde file)

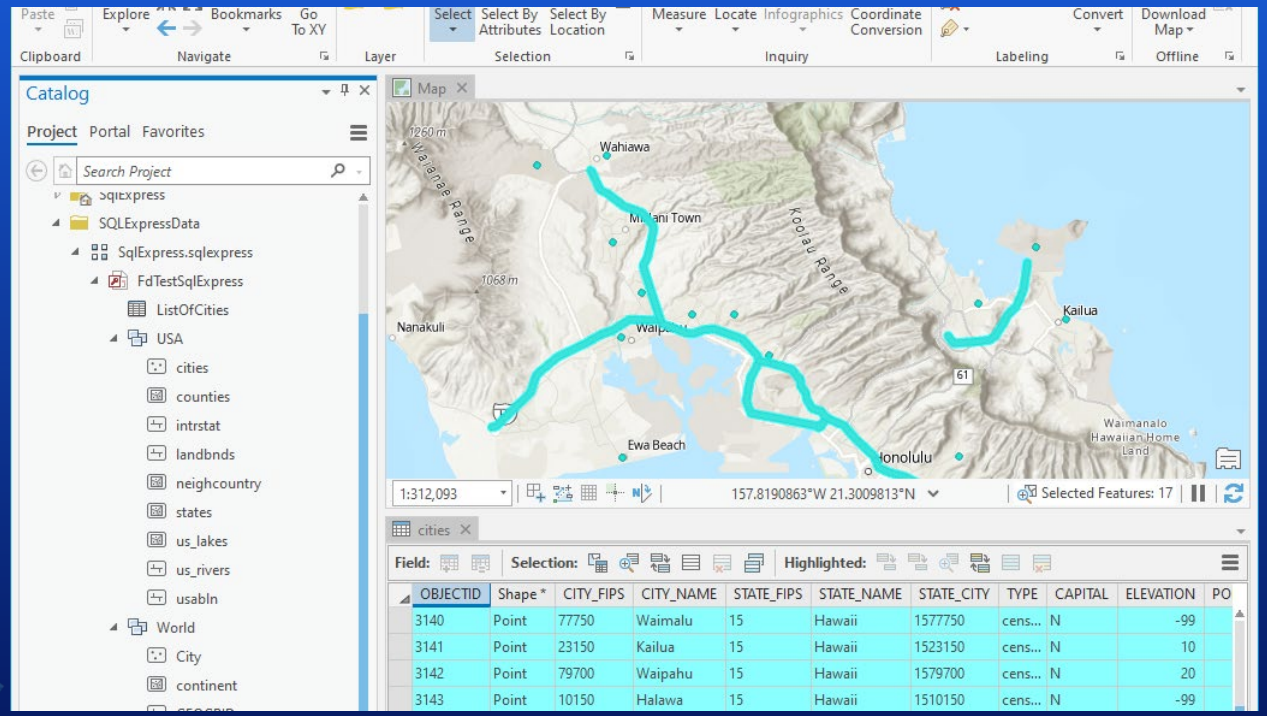
Examples of Custom data source used in Community Samples

- Jpg photos with GPS metadata
  - Smart phones and digital cameras have the option to capture GPS information when a photo is taken
- Gpx File data
  - GPX (the GPS eXchange Format) is a data format for exchanging GPS data between programs and implemented by many GPS tracking devices
- SQL Server Express
  - An example of a free relational database that contains data copied from classic personal geodatabases

Custom data source used In this session: SQL Server Express – custom data source

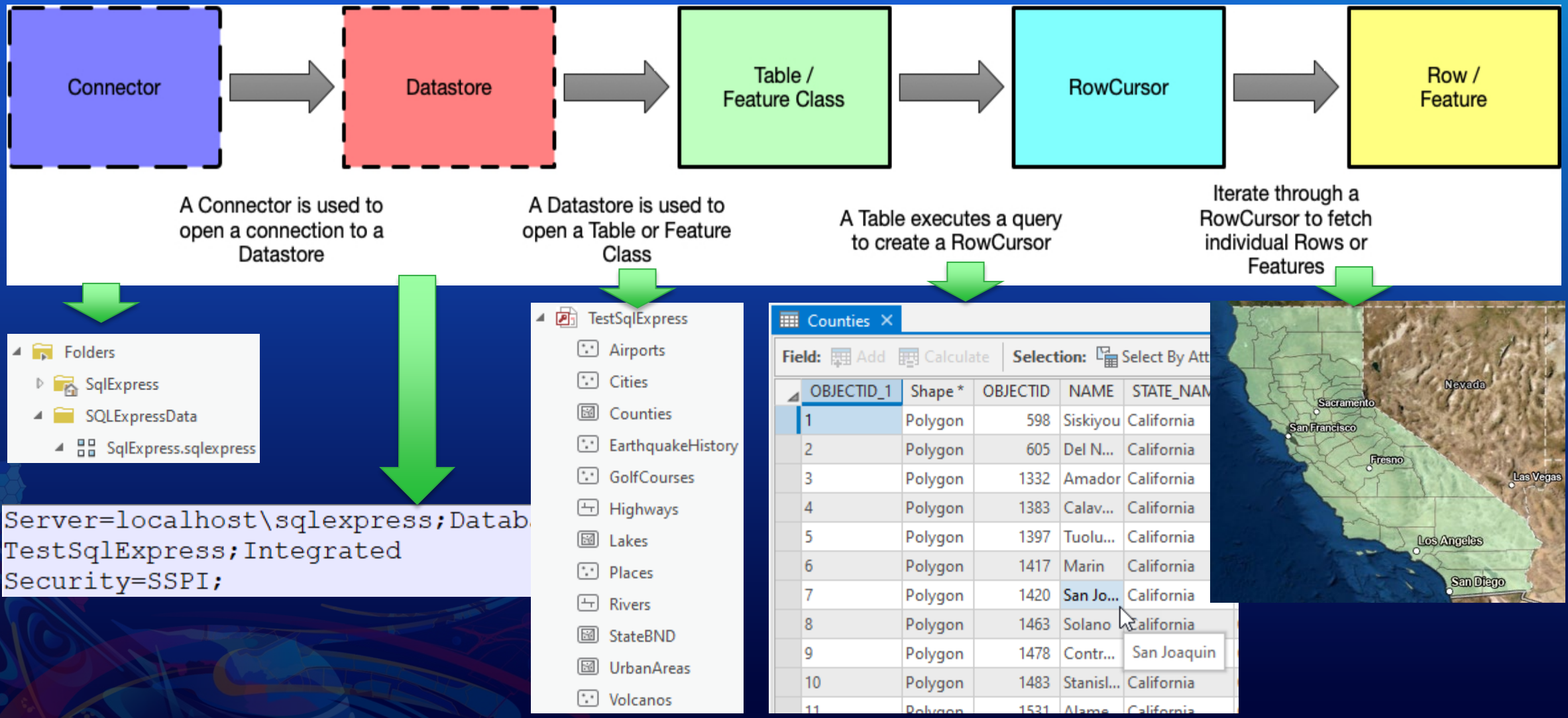
- Instead of creating my own spatial format and data, I copied data from an old ArcMap Personal GDB (Access DB) into my SQLExpress database.
- ArcMap Personal GDBs cannot be directly viewed using Pro, they can only be converted.
- How was this custom data source created?
  - Copied all data tables (containing features and rows) from the Access Database to SQL Express
  - Copied these tables to provide metadata (such as spatial columns, spatial reference, default extent): GDB_GeomColumns, GDB_Items, GDB_SpatialRefs





Demo: Custom (Sql Server Express) Database Plugin Datasource & Project Item

Plugin Datasources: Architecture



Plugin Datasources: Architecture

- ArcGIS Pro Geodatabase functionality is based on the same architecture as the Plugin Datasource Architecture
  - For example: client-server geodatabase such as Oracle, SQL Server, or Postgres.
- How are Plugin Datasources supported by the Pro API?
  - See ArcGIS.Core.Data.PluginDatastore namespace
- The API provides the following template classes:
  - PluginDatasourceTemplate, PluginTableTemplate, and PluginCursorTemplate
  - Implement your custom data source by creating classes that inherit from these template classes
- And a set of supporting classes such as:
  - PluginDatasourceConnectionPath, PluginDatastore, PluginField, and PluginRow

Implementing plugin datasource for your custom data

- Use the ‘ArcGIS Pro Plugin’ Project Template
  - Stubs out all code for you
- Creates three concrete classes that inherit from these API template classes
  - PluginDatasourceTemplate
    - Used to manage one or more tables or feature classes
  - PluginTableTemplate
    - Can be queried and return a cursor
  - PluginCursorTemplate
    - Data Rows are returned by iterating using a cursor
- Update config.daml to register your Plugin Datasource with Pro with the id attribute

```
<ArcGISPro>
  <PluginDatasources>
    <PluginDatasource id="ProSqlPluginDatasource" class="ProSqlPluginDatasourceTemplate" />
  </PluginDatasources>
</ArcGISPro>
```

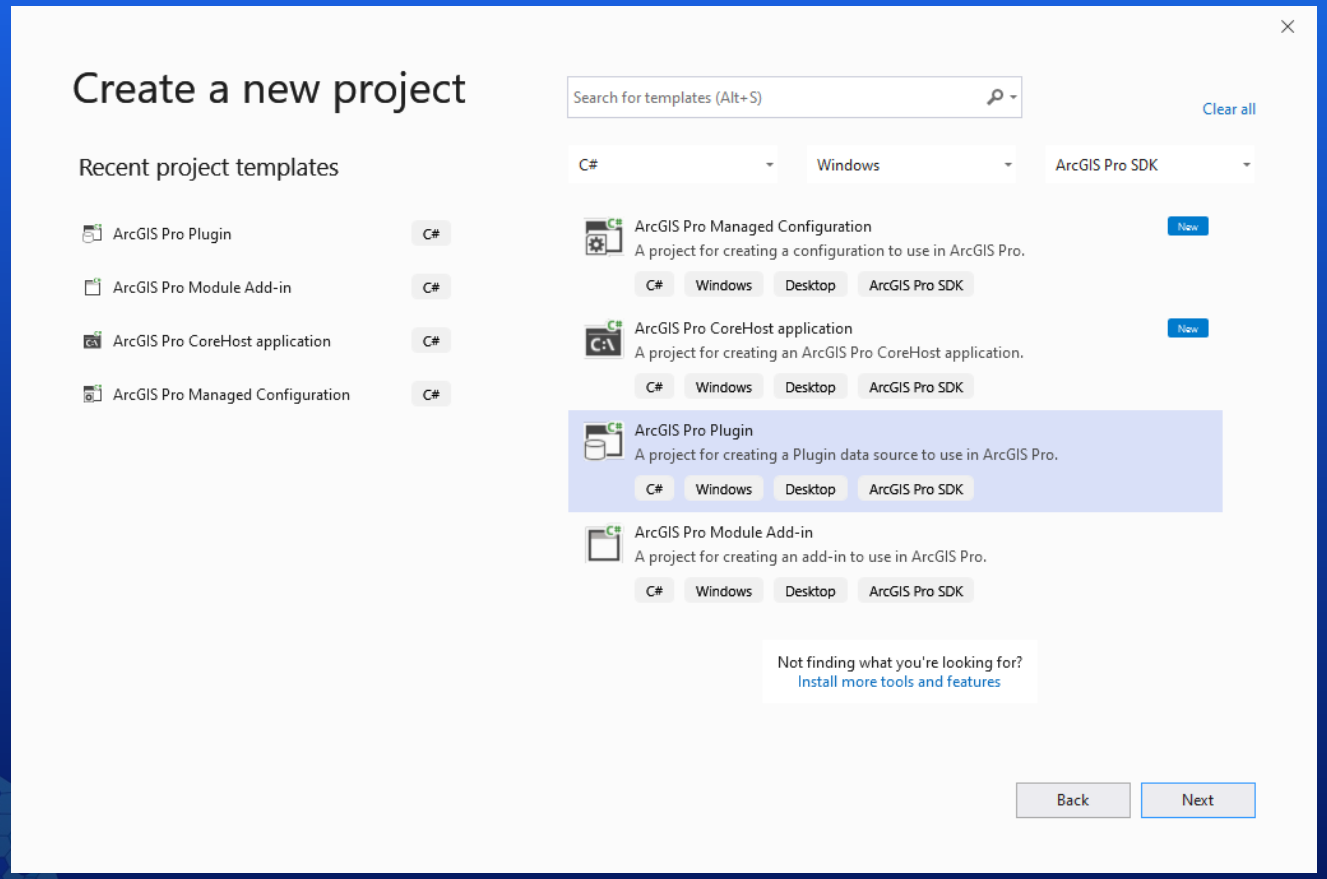
Using a custom plugin datasource in ArcGIS Pro

- How can I use a Custom Plugin Datasource from an Add-in:
  - Create a PluginDatasourceConnectionPath object, which inherits from the Connector base class
  - Create a PluginDatastore object using the connector. PluginDatastore inherits from the Datastore base class
  - Open a Table or FeatureClass using the OpenTable() method on the datastore
  - Using the Table or FeatureClass I can then
  - 1. Extract the data in code by:
    - Using the Search or Select methods to get a RowCursor
    - Iterate through the cursor using MoveNext() and Current, which returns a Row
  - 2. Use the Table or FeatureClass in ArcGIS Pro:
    - - Using LayerFactory to add a FeatureClass to a Map
    - - Using StandaloneTableFactory to add a StandaloneTable to a Map

Using a custom plugin datasource in ArcGIS Pro

- Define a connection to a data source instance for an Add-in
  - Use PluginDatasourceConnectionPath with the PluginDataSource id and the path to the data source file
- Use the connection to create a datastore to your data source instance
- Use the datastore to access tables and feature classes

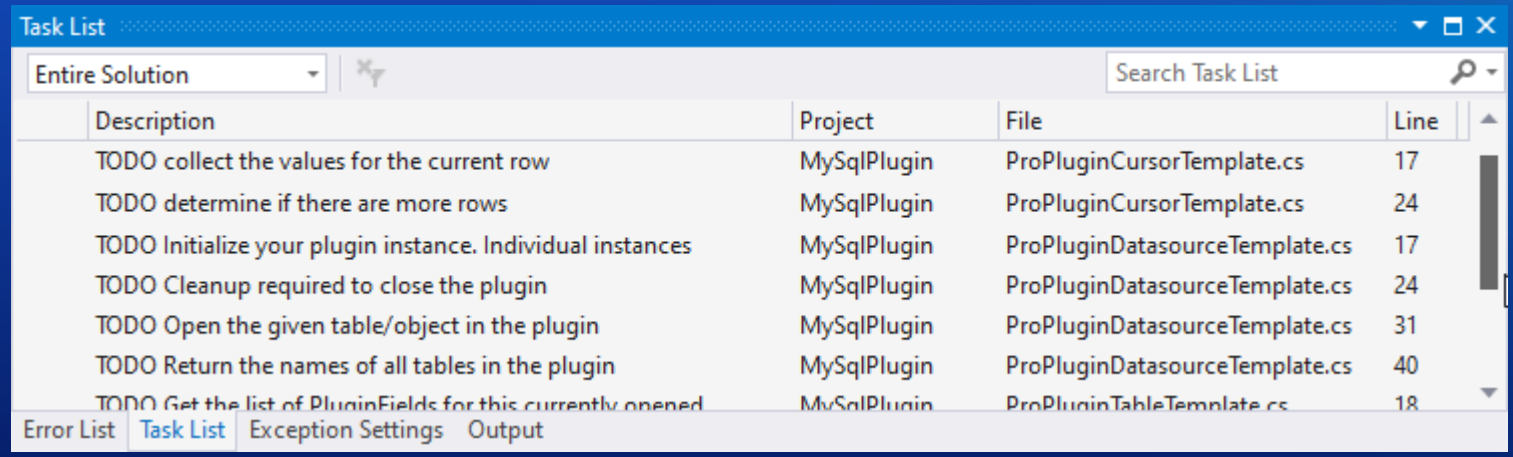
```
var pluginId = @"ProSqlExpressPluginDatasource";
var filePathUri = new Uri(@"C:\Data\PluginData\SQLExpressData\SqlExpress.sqlexpress\\FdTestSqlExpress");
var conSql = new PluginDatasourceConnectionPath(pluginId, filePathUri);
QueuedTask.Run(() =>
{
  using (var pluginSql = new PluginDatastore(conSql))
    foreach (var tableName in pluginSql.GetTableNames())
      using (var tbl = pluginSql.OpenTable(tableName))
        if (tbl is FeatureClass fc)
        {
          //Add as a layer to the active map or scene
          LayerFactory.Instance.CreateFeatureLayer(fc, MapView.Active.Map);
        }
});
```



Demo: Plugin DataSource Template & Addin using the DataSource

“Plugin DataSource Project Template”: Implement Custom Data Access

- Insert your Implementation into the ‘stubbed out’ code of the Project Template for:
  - PluginDatasourceTemplate
  - PluginTableTemplate
  - PluginCursorTemplate
- To navigate required Updates use Visual Studio Task List



Implementing the Plugin Datasource

- PluginDatasourceTemplate
  - public override bool IsQueryLanguageSupported()
    - Return false: The framework filters out rows
    - The framework doesn't relay the user-provided 'where clause' to PluginTable.Search()
  - public override void Open(Uri connectionPath)
    - Access starts file based, so the connectionPath parameter is either a file or folder
    - Sample Uri:
    - file:///C:/Data/PluginData/SQLExpressData/SqlExpress.sqlexpress///FdTestSqlExpress
    - Using '///' to separate the connection file path (.sqlexpress) and the database name

```
Server=localhost\sqlexpress;Database=TestSqlExpress;Integrated Security=SSPI;
Server=localhost\sqlexpress;Database=FdTestSqlExpress;Integrated Security=SSPI;
```

    - Extract the connection properties and connect to the SqlExpress database

Implementing the Plugin Datasource

- PluginDatasourceTemplate
  - public override IReadOnlyList<string> GetTableNames()
    - Using the ‘SqlExpress database’ connection return the list of spatial and non-spatial table names
    - The Plugin Datasource client can use the GetTablesNames to iterate all table names
  - public override PluginTableTemplate OpenTable(string tablePath)
    - tablePath is the name of the table including any dataset paths
    - Sample tablePath: \USA\cities
    - Create and return a ProSqlPluginTableTemplate with SqlExpress connection and the table path as parameters

Implementing the Plugin Datasource

- PluginTableTemplate (Created with SqlExpress connection, table name)
  - public override IReadOnlyList<PluginField> GetFields()
    - Using the SqlExpress connection and table name create a PlugInField list
    - Each PluginField has
      - Name (string)
      - AliasName (string)
      - FieldType (FieldType) which contains the datatype for this field
    - For all tables set one primary key numeric field as FieldType.OID FieldType
    - For spatial tables your shape field has to have a ‘FieldType.Geometry’ FieldType
  - public override GeometryType GetShapeType()
    - Non-spatial tables return a ‘GeometryType.Unknown’ Geometry type
    - Spatial tables return the appropriate GeometryType
      - Example: GeometryType.Point
    - Define the appropriate SpatialReference

Implementing a Plugin Datasource

- PluginTableTemplate
  - public override Envelope GetExtent()
    - Return the spatial extent as an Envelope, using the defined SpatialReference
  - public override PluginCursorTemplate Search(QueryFilter queryFilter)
  - public override PluginCursorTemplate Search(SpatialQueryFilter spatialQueryFilter)
    - Pro Framework calls Search with a queryFilter without any where clauses or spatial filters because IsQueryLanguageSupported() returns false
    - Read all records from the SQLExpress table into a DataTable
    - Create a list of all FieldType.OID values to be used by the cursor
    - Create and return a ProSqlPluginCursorTemplate instance
    - ProSqlPluginCursorTemplate needs the query filter, data table, list of FieldType.OIDs, and the spatial reference (from queryFilter param)

Implementing a Plugin Datasource

- PluginCursorTemplate
  - public override PluginRow GetCurrentRow()
    - Convert the internal row (from DataTable.Rows) to the PluginRow class format
    - Note: the geometry value might have to be reprojected using GeometryEngine.Instance.Project
    - Return the PluginRow
  - public override bool MoveNext()
    - The list of all FieldType.OID values is used to advance the cursor to the next position
    - MoveNext () is called to get to the first record and to advance the cursor to the next position
    - Returns false if no records are left to return, true if the cursor was positioned over another record.

Supporting Queries (where clause) in a Plugin Datasource

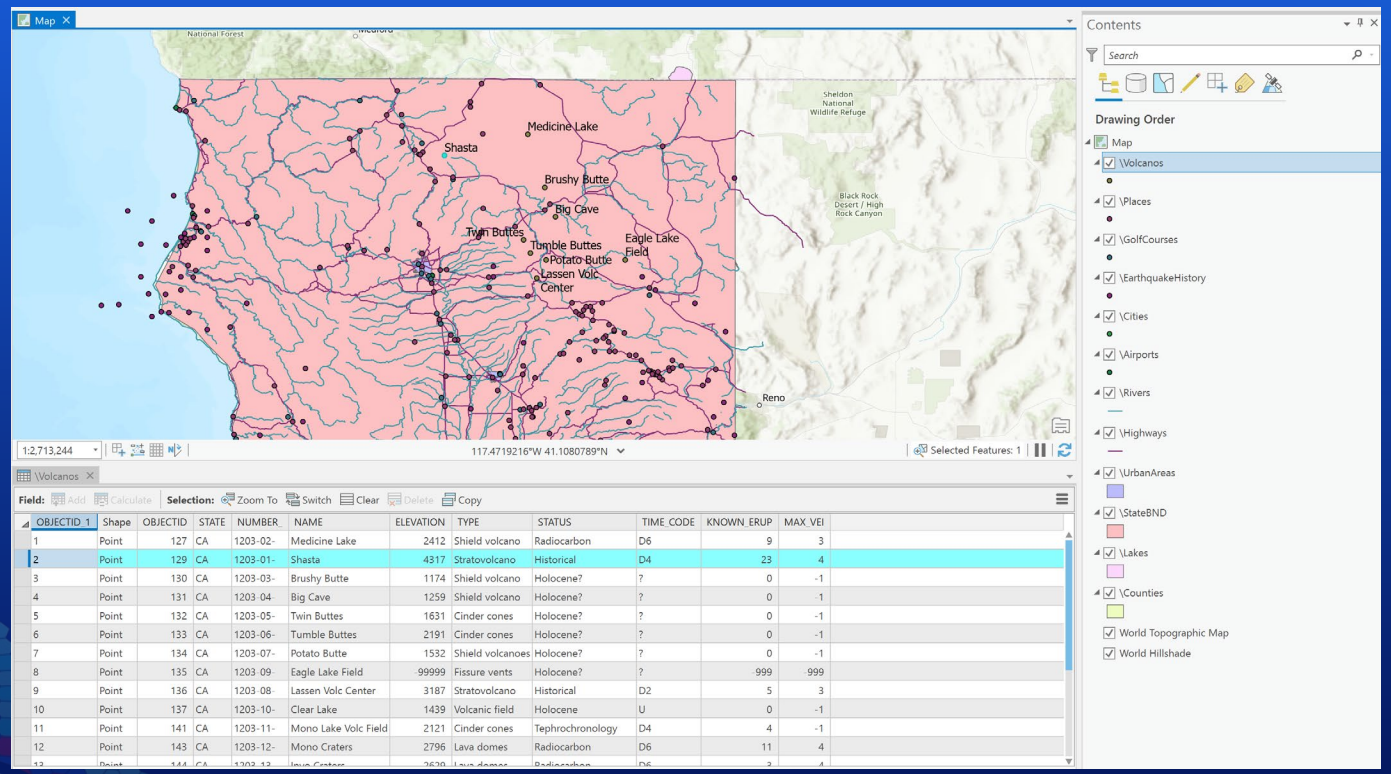
- PluginDatasourceTemplate
  - public override bool IsQueryLanguageSupported()
    - Return true, the Pro framework passes a where clause into Table.Search() which passes it directly to PluginTable.Search() (both spatial and non-spatial queries)

- PluginTableTemplate
  - public override PluginCursorTemplate Search(QueryFilter queryFilter)
  - public override PluginCursorTemplate Search(SpatialQueryFilter spatialQueryFilter)
    - Process any where clauses first (this includes the List of ObjectIds)
    - DataTable supports where clauses and order by clauses … so no custom code here
    - For spatial queries use where clause result to run a spatial filter on each shape
      - Convert your ‘custom’ shape format into a valid Pro API Geometry first
    - Return the List of ObjectIds (in the cursor) for all records matching the query filter

Supporting Queries (where clause) in a Plugin Datasource

- **Implementing a spatial filter to check each record's geometry:**
  - **Returns false: the record geometry doesn't pass spatial filter requirements**

```
internal static bool HasRelationship(Geometry geomSpatialFilter,
                            Geometry geomRecordToCheck,
                            SpatialRelationship relationship)
{
    var engine = GeometryEngine.Instance;
    switch (relationship)
    {
         case SpatialRelationship.Intersects:
              return engine.Intersects(geomSpatialFilter, geomRecordToCheck);
         case SpatialRelationship.IndexIntersects:
              return engine.Intersects(geomSpatialFilter, geomRecordToCheck);
…..
         case SpatialRelationship.Within:
              return engine.Within(geomSpatialFilter, geomRecordToCheck);
    }
    return false;//unknown relationship
}
```



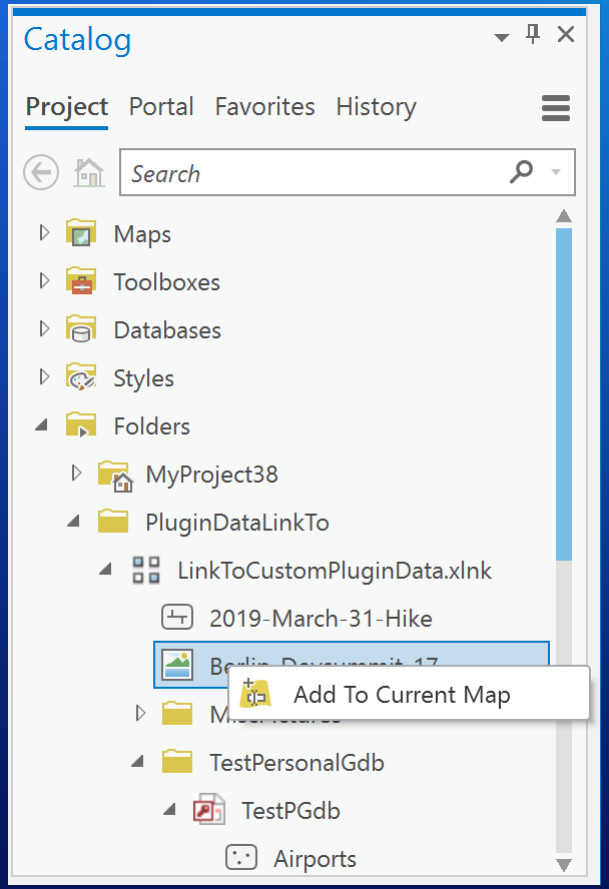
OBJECTID_1	Shape	OBJECTID	STATE	NUMBER_	NAME	ELEVATION	TYPE	STATUS	TIME_CODE	KNOWN_ERUP	MAX_VEI
1	Point	127	CA	1203-02-	Medicine Lake	2412	Shield volcano	Radiocarbon	D6	9	3
2	Point	129	CA	1203-01-	Shasta	4317	Stratovolcano	Historical	D4	23	4
3	Point	130	CA	1203-03-	Brushy Butte	1174	Shield volcano	Holocene?	?	0	-1
4	Point	131	CA	1203-04-	Big Cave	1259	Shield volcano	Holocene?	?	0	-1
5	Point	132	CA	1203-05-	Twin Buttes	1631	Cinder cones	Holocene?	?	0	-1
6	Point	133	CA	1203-06-	Tumble Buttes	2191	Cinder cones	Holocene?	?	0	-1
7	Point	134	CA	1203-07-	Potato Butte	1532	Shield volcanoes	Holocene?	?	0	-1
8	Point	135	CA	1203-09-	Eagle Lake Field	-99999	Fissure vents	Holocene?	?	-999	999
9	Point	136	CA	1203-08-	Lassen Volc Center	3187	Stratovolcano	Historical	D2	5	3
10	Point	137	CA	1203-10-	Clear Lake	1439	Volcanic field	Holocene	U	0	-1
11	Point	141	CA	1203-11-	Mono Lake Volc Field	2121	Cinder cones	Tephrochronology	D4	4	-1
12	Point	143	CA	1203-12-	Mono Craters	2796	Lava domes	Radiocarbon	D6	11	4

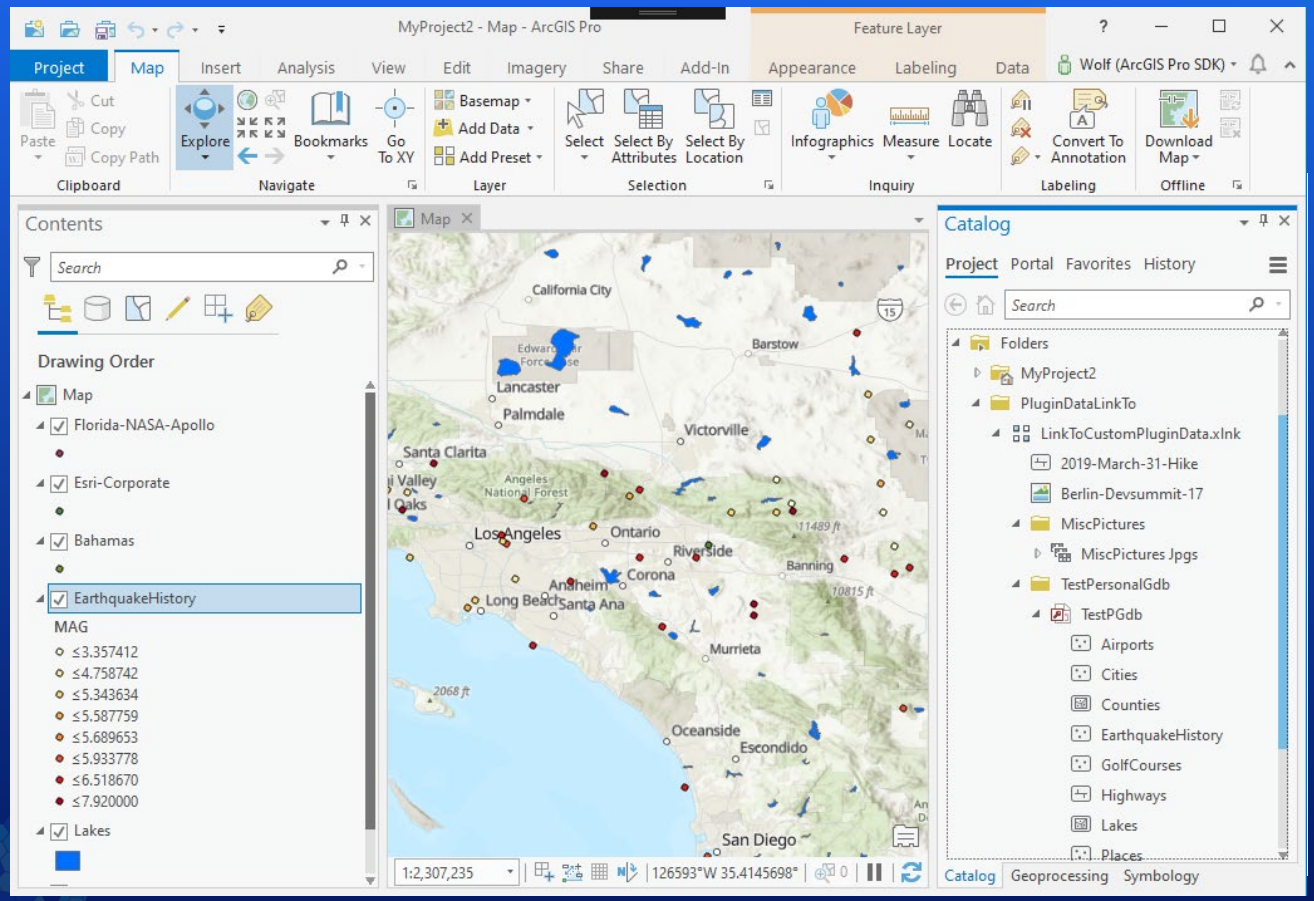
Demo: Plugin DataSource Sample

C:\Data\PluginData\SQLExpressData\SqlExpress.sqlexpress

Better integration of custom plugin datasource in ArcGIS Pro

- **A better user experience is:**
  - Custom content is integrated both into the Pro catalog and its browse experience
- **Use the 'Custom Project Item' item template in your Add-in**
  - Pairing a custom item with a plugin datasource allows your custom content to be integrated into the Pro catalog and open file dialog
- **Use this sample implementation as a guide**
- **Use ProConcepts and ProGuide Custom Items**





Demo: Plugin DataSource Sample

Plugin Datasources, Deep Dive

- **Session content**
  - **Slides and Example code:**
  - **https://github.com/esri/arcgis-pro-sdk/wiki/tech-sessions**