ArcGIS Pro SDK for .NET: An Overview of the Geodatabase API

Rich Ruh, Alex Nguyen, Ling Zhang
Today’s Agenda

- Architecture Principles
- Overview of the Geodatabase API
- What’s New
- Best Practices
- Road Ahead
- Learning More
What We Won’t Cover

• Add-in model
• Editing
• Detailed comparison of ArcObjects vs. Pro SDK
Architectural Principals

Alex Nguyen
Core.Data – a geodatabase C# API powered by ArcObjects
Key API Characteristics

- Designed and implemented using industry standard C# patterns and idioms
- It is minimally complete – exposes essential abstractions in only a few namespaces:
  - ArcGIS.Core.Data and sub namespaces, e.g., ArcGIS.Core.Data.Raster
- Provides a strongly typed object model
  - Minimizes object type casting
  - Helps make the API discoverable and easy to use correctly
- Offers good support for exception handling
  - ArcGIS.Core.Data.GeodatabaseGeneralException: Cannot acquire a lock. versus
    System.Runtime.InteropServices.COMException: Exception from HRESULT: 0x8004022D in ArcObjects
Data Manipulation Language Only

- Core.Data is a DML-only (Data Manipulation Language) API
  - Cannot perform schema creation or modification operations:
    - creating tables or adding fields
    - creating domains or adding coded values
  - Schema operations are performed using the GP (Geoprocessing) tools
  - GP tools can be called from C# using the Geoprocessing API
    (ArcGIS.Desktop.Core.Geoprocessing namespace)

```csharp
IReadOnlyList<string> args = Geoprocessing.MakeValueArray(@"C:\Data.gdb", "Places");
IGPResult gpResult = await Geoprocessing.ExecuteToolAsync("CreateFeatureclass_management", args);
if (gpResult.IsFailed)
{
    // ...
}
```
Threading

• Almost all of the methods in Core.Data API should be called on the MCT (Main CIM Thread)
  - API reference documentation on the methods that need to run on the MCT are specified
  - These methods calls should be wrapped inside the QueuedTask.Run call
  - Failure to do so will result in CalledOnWrongThreadException or ConstructedOnWrongThreadException being thrown

• Read “Working with multithreading in ArcGIS Pro” conceptual help to learn more

```csharp
await QueuedTask.Run(() =>
{
    using (Geodatabase geodatabase = new Geodatabase(new DatabaseConnectionFile(new Uri(sdeFilePath))))
    {
        workspaceConnectionString = geodatabase.GetConnectionString();

        // ...
    }
});
```
Overview of the Geodatabase API
Overview of the Geodatabase API

- Datastores and Datasets
- Queries
- Versioning
- CoreHost
- Plug-in Datasources
- Attribute Rules
Datastores and Datasets
General Geodatabase Access Pattern

Connector -> Datastore -> Table / Feature Class -> RowCursor -> Row / Feature

- A Connector is used to open a connection to a Datastore.
- A Datastore is used to open a Dataset.
- Tables and Feature Classes execute queries to create RowCursors.
- Iterate through a RowCursor to fetch individual Rows or Features.
Connectors

A Connector is used to open a connection to a Datastore.
Datastore and Geodatabase

- Container of spatial and non-spatial datasets
- Created by a Connector
- Can also be obtained from a dataset
  - Dataset.GetDatastore()
Types of Datastores

- Geodatabase
  - Enterprise Geodatabases (Oracle, SQL Server, Postgres, Hana)
  - File Geodatabases
  - Feature Service Geodatabases
- FileSystemDatastore
  - Shape files
- Database
  - DBMS without geodatabase enabled
Datasets — Table

- Table
  - Contains 0 or more Rows
  - Supports Search and Select
  - `geodatabase.OpenDataset<Table>("TableName");`
  - `featureLayer.GetTable();`
  - `row.GetTable();`
Datasets — Feature class

- **Feature class**
  - Inherit from tables
  - Tables with shape (point, line, polygon)
  - Contains 0 or more Features
  - Supports spatial queries and Selections

  ```csharp
  geodatabase.OpenDataset<FeatureClass>("FeatureClassName");
  featureLayer.GetTable() as FeatureClass;
  row.GetTable() as FeatureClass;
  Open as table:
  Geodatabase.OpenDataset<Table>("FeatureClassName");
  ```
Datasets — Feature dataset

• Feature Dataset
  - Collection of related datasets with a common coordinate system
  - Organize feature classes to support topology, network dataset, utility network, or terrain dataset.
  - Acts as a container

- `geodatabase.OpenDataset<FeatureDataset>("FeatureDatasetName")`
Datasets — Relationship Classes

- Geodatabase stored relationship between tables/feature classes
- Origin and Destination tables
- Cardinality of relationship between features

- Two types:
  - RelationshipClass
    - May not have a backing table
  - AttributedRelationshipClass
    - Inherits from RelationshipClass
    - Has a backing table
    - May have user defined attributes
Definitions

- Concept to represent information about the dataset
- Used to describe schema and unique properties
  - e.g., GetFields, HasGlobalID
- Separated from the dataset to facilitate a lightweight mechanism of discovery
- Opening dataset comparatively expensive
- Definitions can be used to filter datasets without opening them

```csharp
table.GetDefinition();
geodatabase.GetDefinition<TableDefinition>("TableName");
geodatabase.GetDefinitions<TableDefinition>();
```
Queries
Different Techniques to Execute Queries

- Simple search
- QueryDef
- QueryTable
- Join
- QueryLayer
Basic Queries using Search and Select

- **Search**
  - `table.Search()`
  - Return Rows (or subset of values) via RowCursor
  - Supports Recycling

- **Selection**
  - `table.Select()`
  - List of Object IDs
  - Lightweight way to highlight features on map
  - Ability to combine
Querying Data — Filtering queries

- **QueryFilter**
  - Used to restrict the records retrieved
    - Where clause
    - Subfields
  - Apply pre- or postfix clauses
    - DISTINCT
    - ORDER BY

- **SpatialQueryFilter**
  - Restrict retrieve records using a spatial query
Row Cursor

- Represents a cursor
- Allows you to walk through a collection of rows

- **MoveNext()**
  - Positions the cursor at the next row in the sequence
  - Returns false when finished

- **Current**
  - Returns the current row
Querying Data — QueryDef

• QueryDef
  - Available with file and enterprise geodatabases
  - Tables are input parameter
    - Single table query
    - Or two or more joined tables within the same datastore
  - Rows do not implement GetTable()
  - Does not support field aliases
  - ‘Left’ most shape field supported
  - Result is read-only

  - geodatabase.Evaluate(queryDef, false);
const string PRIMARY_KEY_IN_PARCELS_FEATURE_CLASS = "PROPERTY_I";
const string FOREIGN_KEY_IN_OWNERS_TABLE = "PROPERTY_ID";
QueryDef queryDef = new QueryDef()
{
    Tables = String.Format("{0} INNER JOIN {1} ON {2} = {3}",
                          "Parcels", "Owners",
                          PRIMARY_KEY_IN_PARCELS_FEATURE_CLASS,
                          FOREIGN_KEY_IN_OWNERS_TABLE),
    PostfixClause = "ORDER BY OWNER_NAME"
};
Querying Data — QueryTable

- **QueryTable**
  - Read-only
  - Virtual table (for map layer or Geoprocessing)
  - Requires a **QueryTableDescription**
    - Name
    - QueryDef
    - Key fields (Object ID / Shape)

- `geodatabase.OpenQueryTable(QueryTableDescription);`
Querying Data — Joins

- Used to combine fields from two tables into a single table representation
- Supports tables from one or two datastores
- Created using relationship classes
  - Relationship class stored in the geodatabase
  - Or virtual relationship class
- Result is read-only, but reflects updates to underlying tables
- Does not support many-many cardinality
Joins — Virtual Relationship Classes

- Used to create a Join between tables in different datastores
  1. Create a `VirtualRelationshipClassDescription`
  2. `table.RelateTo(rightTable, virtualRelationshipClassDescription)` creates a virtual relationship class
  3. Pass that to the `JoinDescription(RelationshipClass)` constructor
  4. Pass that to the `Join(JoinDescription)` constructor
Joins

- `join.GetJoinedTable()`
- Definitions not supported on joined tables where tables are from multiple datastores
- Every attempt will be made to push join to database for performance
  - Within a single datastore
Databases and QueryLayers

- Database: Datastore for a DBMS without geodatabase enabled
- Allows queries of these tables via QueryLayers

- QueryLayers are created via `Database.GetQueryDescription`
  - From a table
  - From a table name
  - From a string containing a SQL query
- Modify `QueryDescription`
  - unique ID field, must have not Nulls
  - single geometry column
- Pass to `Database.OpenTable(QueryDescription)`
Different Techniques to Execute Queries

- Simple search
- QueryDef
- QueryTable
- Join
- QueryLayer
Some Caveats…

- Feature service databases have additional restrictions
  - Don’t work with QueryDef
  - Don’t work with Join

- Joins are simpler; QueryDefs have more capabilities
Sorting Tables

- Create a row cursor in a sorted order
- `Table.Sort(TableSortDescription)`
Calculating Statistics

- Calculate statistics on table fields
  - Count, Sum, Min, Max, Average, Standard Deviation
  - Including support for Group By and Order By
- Computed on the server if supported
Calculating Statistics

- `Table.CalculateStatistics()` works on the table or feature class, not the layer
- ArcGIS Pro UI and geoprocessing work on the layer, not the feature class
Datastores and Datasets

Ling Zhang
Calculating Statistics
Ling Zhang
Versioning
Versioning Support

- **VersionManager**
  - Available only if `Geodatabase.IsVersioningSupported()` returns True
  - Obtained with `Geodatabase.GetVersionManager()`

- **Current capabilities**
  - Create Versions
  - Connect to a specific named version
  - List all versions in a geodatabase, including properties
  - List differences between Tables and Feature Classes from different Versions (`Table.Differences()`)

**VersionManager**

```
Methods
- GetVersionManager() : VersionManager
- IsVersioningSupported() : bool
```

**HistoricalVersion**

```
Properties
- AccessType (get; set) : VersionAccessType
- Description (get; set) : string
- Name (get; set) : string
```

**VersionDescription**

```
Properties
- AccessType (get; set) : VersionAccessType
- Description (get; set) : string
- Name (get; set) : string
```

**Methods**

```
- VersionDescription()
```
Versioning Support

- Version
  - Alter or Delete
  - Reconcile and Post
  - Refresh
Core.Host

- Using CoreHost in ArcGIS.Core.Hosting namespace
  - Programs using the API can be run “head-less” (i.e., standalone)
  - GP tools must be run directly from Python

- Requirements
  - References (ArcGIS.Core & ArcGIS.CoreHost)
  - 64-bit app with [STAThread]
  - Host.Initialize()
  - Licensing controlled by Pro application

- ArcGIS.Desktop.Editing not possible
  - Geodatabase.ApplyEdits()
    - Not to be used in Add-in
Plug-in Datasources
Plug-in Datasources

- ArcGIS Pro can display and analyze data from many different formats
- Many other data formats are not supported
  - Other databases such as MySQL
  - Proprietary or obscure file-based data sources
- Plugin Datasource framework allows developers to make these and other data formats available to ArcGIS Pro
- Execute queries, perform analysis, and display features on a map
- Read-only access only
Plug-in Datasources

- New Connector object: `PluginDatasourceConnectionPath`
- Framework provides set of template classes
- Developers fill in implementation-specific pieces for datastores, tables, and cursors
Attribute Rules
Attribute Rules

- Attribute rules enhance the editing experience and improve data integrity
- User-defined rules written using Arcade
- Assigned to feature classes and subtypes using geoprocessing tools

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>Restrict invalid edits</td>
<td>At edit time</td>
</tr>
<tr>
<td>Calculation</td>
<td>Populate attributes</td>
<td>At edit time</td>
</tr>
<tr>
<td>Validation</td>
<td>Perform quality assurance checks on existing features</td>
<td>Batch</td>
</tr>
</tbody>
</table>
Types of Attribute Rules

- These rules fire when edits take place
- Supported on all enterprise geodatabases
- No explicit SDK support (they just work)

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>Restrict invalid edits</td>
<td>At edit time</td>
</tr>
<tr>
<td>Calculation</td>
<td>Populate attributes</td>
<td>At edit time</td>
</tr>
<tr>
<td>Validation</td>
<td>Perform quality assurance checks on existing features</td>
<td>Batch</td>
</tr>
</tbody>
</table>
Types of Attribute Rules

- These rules fire by calling an evaluation process
- Supported on feature service workspaces using branch versioning
  - Requires ArcGIS Enterprise 10.7
- Evaluation process can be called from the SDK

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>Restrict invalid edits</td>
<td>At edit time</td>
</tr>
<tr>
<td>Calculation</td>
<td>Populate attributes</td>
<td>At edit time</td>
</tr>
<tr>
<td>Validation</td>
<td>Perform quality assurance checks on existing features</td>
<td>Batch</td>
</tr>
</tbody>
</table>
Attribute Rules — Evaluation

• Attribute Rule Manager obtained from Geodatabase
  - IsAttributeManagerSupported()
  - GetAttributeRuleManager()

• Evaluation called from the Attribute Rule Manager
  - IsEvaluationSupported()
  - Evaluate()

• Additional SDK capabilities
  - Fetch error features
  - Fetch original features that generated errors
  - Mark errors as exceptions
What’s New
ArcGIS Pro 2.4 and 2.5
Historical Versions

- Read-only representation of geodatabase at a specific time
  - Only applies to archived datasets
- Create an historical marker using `CreateHistoricalVersion`
  - “Twin Pines Mall safety incident”
  - 1:15 am on October 26, 1985
- Connect to a specific time or an historical marker (`GetHistoricalVersion`)
Geodatabase Grab Bag

- Provide extension methods to evaluate attribute rules (EvaluateInEditOperation) and update errors (UpdateErrorsInEditOperation) inside an edit operation
- Added a constructor for VersionDescription
- Provide access to the Feature Dataset from a FeatureClass or Relationship Class (GetFeatureDataset())
- Improvements to Plug-in Datasource APIs
Best Practices

Alex Nguyen
Garbage Collection

• By default, locks and connections on database objects held by .NET objects are released at non-deterministic times (When garbage collection runs)
  - As a consequence, connections and locks on any dataset referenced by the .NET object will also be held

• Using blocks or calling Dispose on objects after use provides greater control on when connections and file system locks are released

```csharp
FeatureClass featureClassIndeterministic = geodatabase.OpenDataset<FeatureClass>(featureClassName);
using (FeatureClass featureClassDeterministic = geodatabase.OpenDataset<FeatureClass>(featureClassName))
{
    Console.WriteLine(featureClassDeterministic.GetName());
}
```

• Locks acquired by featureClassIndeterministic are released when garbage collector runs
• Locks acquired by featureClassDeterministic are released at the end of the using block
Best Practices

- Do explicitly dispose of objects of the following types:
  - `ArcGIS.Core.Data.Datastore` and its derived classes (e.g., `Geodatabase`)
  - `ArcGIS.Core.Data.Dataset` and its derived classes (e.g., `FeatureDataset`, `Table`, `FeatureClass`, `RelationshipClass`, etc)
  - `ArcGIS.Core.Data.RowCursor`
  - `ArcGIS.Core.Data.RowBuffer`
  - `ArcGIS.Core.Data.Row` and its derived class, `Feature`
  - `ArcGIS.Core.Data.Selection`
  - `ArcGIS.Core.Data.VersionManager`
  - `ArcGIS.Core.Data.Version`
  - `ArcGIS.Core.AttributeRuleManager`
Best Practices (cont’d)

• Two standard ways to explicitly dispose of objects in C#
  - Via the **try/finally** block:

```csharp
public void DoSomething()
{
    Geodatabase geodatabase = null;
    FeatureClass featureClass = null;

    try
    {
        geodatabase = new Geodatabase(connectionProperties);
        featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS);
        //...
    }
    finally
    {
        if (featureClass != null)
            featureClass.Dispose();

        if (geodatabase != null)
            geodatabase.Dispose();
    }
}
```
Best Practices (cont’d)

- Via the *using* statement:

```csharp
public void DoSomething()
{
    using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
    {
        using (FeatureClass featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS))
        {
            //...
        }
    }
}
```

```csharp
public void DoSomething()
{
    using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
    {
        using (FeatureClass featureClass = geodatabase.OpenDataset<FeatureClass>(SOME_FEATURE_CLASS))
        {
            //...
        }
    }
}
```
Best Practices (cont’d)

- Do remember to explicitly dispose of Row/Feature returned from RowCursor
  - Not explicitly disposing of Row/Feature:

```csharp
public static long CountRows(Table table) {
    using (RowCursor rowCursor = table.Search()) {
        long count = 0;
        while (rowCursor.MoveNext()) {
            ++count;
        }
        return count;
    }
}
```
Best Practices (cont’d)

- Explicitly disposing of Row/Feature:

```csharp
public static long CountRows(Table table)
{
    using (RowCursor rowCursor = table.Search())
    {
        long count = 0;

        while (rowCursor.MoveNext())
        {
            ++count;
            rowCursor.Current.Dispose(); // Need to call Dispose on the Row/Feature as well.
        }

        return count;
    }
}
```
Best Practices (cont’d)

- Canonical form of processing a `RowCursor`:

```csharp
public void DoSomething()
{
    using (Geodatabase geodatabase = new Geodatabase(connectionProperties))
    using (Table table = geodatabase.OpenDataset<FeatureClass>(SOME_TABLE))
    {
        using (RowCursor rowCursor = table.Search())
        {
            while (rowCursor.MoveNext())
            {
                using (Row row = rowCursor.Current)
                {
                    //...
                }
            }
        }
    }
}
```
Best Practices (cont’d)

• Don’t create static variables for classes derived from `ArcGIS.Core.CoreObjectBase`.

```java
public class SomeClass
{
    private static Table table = Factory.InitializeTable();

    //...
}
```

- will cause `ConstructedOnWrongThreadException` because all static variables will be executed on the GUI thread when an add-in is first loaded by ArcGIS Pro.
The Road Ahead
Near Term

• Versioning API enhancements
  - Get versioning type (branch or traditional)
  - Get version names
  - Exposing reconcile ConflictDetectionType and ConflictResolutionType
  - Change owner of a version

• `GetPath()` on Dataset

• Feature Cursor support
  - Fast inserts in CoreHost application
Long Term

- DDL
- Topology
Learning More
Conceptual Doc

- Or just search for “ArcGIS Pro SDK”
Conceptual Doc

ProConcepts Geodatabase

Functionality that uses the fine-grained Geodatabase API. Geodatabase API functionality is found in ArcGIS.Core.dll. The Geodatabase API is commonly used in conjunction with map exploration, map authoring, and editing.

Language: C# and Visual Basic
Subject: Geodatabase
Contributor: ArcGIS Pro SDK Team <arcgisprosdk@esri.com>
Date: 1/14/2019
ArcGIS Pro: 2.3
Visual Studio: 2015, 2017

In this topic

- Architecture
  - Resource Management
- Dataset
  - Geodatabase
    - QueryDef
    - QueryTable
  - Database
    - Query layers
- Home
  - Samples
  - ProSnippets
- Developing with ArcGIS Pro
  - Requirements
  - Installing ArcGIS Pro SDK for .NET
  - Getting started
  - ProGuide: ArcGIS Pro
    - Extensions NuGet
    - ProConcepts: Migrating to ArcGIS Pro
  - ProSnippets
  - ArcGIS Pro API
  - Release notes
  - Resources
ProConcepts Geodatabase

Functionality that uses the fine-grained Geodatabase API. Geodatabase API functionality is found in ArcGIS.Core.dll. The Geodatabase API is commonly used in conjunction with map exploration, map authoring, and editing.

Language: C# and Visual Basic
Subject: Geodatabase
Contributor: ArcGIS Pro SDK Team <arcgisprosdk@esri.com>
Date: 1/14/2019
ArcGIS Pro: 2.3
Visual Studio: 2015, 2017

In this topic
- Architecture
  - Resource Management
- Datastore
  - Geodatabase
    - QueryDef
    - QueryTable
  - Database
    - Query layers
### Samples

- Suggestions?
- Write your own?
ProConcepts Geodatabase

Functionality that uses the fine-grained Geodatabase API. Geodatabase API functionality is found in ArcGIS.Core.dll. The Geodatabase API is commonly used in conjunction with map exploration, map authoring, and editing.

Language: C# and Visual Basic
Subject: Geodatabase
Contributor: ArcGIS Pro SDK Team <arcgisprosdk@esri.com>
Date: 1/14/2019
ArcGIS Pro: 2.3
Visual Studio: 2015, 2017

In this topic

- Architecture
  - Resource Management
- Datastore
  - Geodatabase
    - QueryDef
    - QueryTable
  - Database
    - Query layers

Developing with ArcGIS Pro

- Requirements
- Installing ArcGIS Pro SDK for .NET
- Getting started
- ProGuide: ArcGIS Pro Extensions NuGet
- ProConcepts: Migrating to ArcGIS Pro
- ProSnippets
- ArcGIS Pro API
- Release notes
- Resources
Snippets

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

Searching a Table using a set of ObjectIDs

```java
public RowCursor SearchingATable(Table table, IReadonlyList<long> objectIDs)
{
    QueryFilter queryFilter = new QueryFilter()
    {
        ObjectIDs = objectIDs
    }

    return table.Search(queryFilter);
}
```
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.
API Reference

ArcGIS.Core.Data Namespace (ArcGIS.Core)

The ArcGIS.Core.Data namespace contains types that support data access.

### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachment</td>
<td>Represents a document associated with a row.</td>
</tr>
<tr>
<td>AttributedRelationship</td>
<td>Represents a pair of related rows (or features) from a geodatabase with extra information stored as a row in an intermediate table.</td>
</tr>
<tr>
<td>AttributedRelationshipClass</td>
<td>Represents an association between two tables in a geodatabase on which additional information is stored in an intermediate table.</td>
</tr>
<tr>
<td>AttributedRelationshipClassDefinition</td>
<td>Represents the schema and properties of an AttributedRelationshipClass.</td>
</tr>
<tr>
<td>AttributeRuleError</td>
<td>Represents an error from a ValidationErrortype validation error system table.</td>
</tr>
<tr>
<td>AttributeRuleEvaluationResult</td>
<td>Provides information about the results of a call to AttributeRuleManager.Evaluate.</td>
</tr>
<tr>
<td>AttributeRuleManager</td>
<td>This class provides basic functionality to manage attribute rules on behalf of a geodatabase.</td>
</tr>
<tr>
<td>CodedValueDomain</td>
<td>Represents the information about the valid coded values belonging to this coded value domain.</td>
</tr>
<tr>
<td>Connector</td>
<td>Represents a connector to a data source.</td>
</tr>
<tr>
<td>Database</td>
<td>Represents the gateway to connect to a relational database, which may or may not be a geodatabase (please see remarks). Given a specific QueryDescription object, the Database data store can be used to open a single database table, or a query layer that is created from one or more database tables.</td>
</tr>
<tr>
<td>DatabaseClient</td>
<td>Provides a capability to execute tasks (e.g., a SQL statement) on a database management system associated with a geodatabase.</td>
</tr>
<tr>
<td>DatabaseConnectionFile</td>
<td>Represents the physical path to a connection file that ends with the .sde extension used for connecting to an enterprise database or geodatabase.</td>
</tr>
<tr>
<td>DatabaseConnectionProperties</td>
<td>Represents the properties used to connect to an enterprise database or geodatabase.</td>
</tr>
<tr>
<td>Dataset</td>
<td>Represents a dataset from a specific Dataset.</td>
</tr>
</tbody>
</table>
An Overview of the Geodatabase API

• Session materials are available here:
  