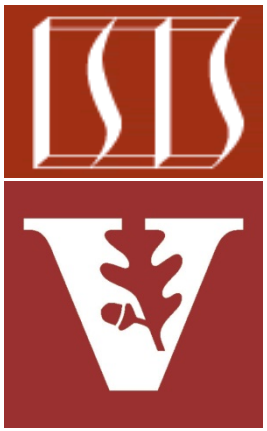
Android Services & Local IPC: Advanced Bound Service Communication – AIDL Syntax & Supported Data Types

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the Android interface syntax & supported data types

```
InterfaceDeclaration:
    interface Identifier InterfaceBody
InterfaceBody:
    { { InterfaceBodyDeclaration } }
InterfaceBodyDeclaration:
    InterfaceMethodDecl
InterfaceMethodDecl:
    Type Identifier InterfaceMethodDeclaratorRest
InterfaceMethodDeclaratorRest:
    FormalParameters
```

```
                        interface IDownloadCallback {
                          oneway void sendPath(in String path);
                        }
interface Idownload {
    oneway void setCallback(in IDownloadCallback callback);
}
```

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

frameworks/base/core/java/com/android/internal/os/IDropBoxManagerService.aidl

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their “business” logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
    - AIDL can declare methods with typed parameters & a return value

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
  - Differences from Java interfaces
    - No static fields

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
  - Differences from Java interfaces
    - No static fields
    - All non-primitive parameters must be labeled by "direction"
      - **`in`** – (default/only mode for primitives) transferred to remote method
      - **`out`** – returned to the caller
      - **`inout`** – both in & out (rarely used)

Limit direction to just what's needed since marshaling parameters is expensive

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
  - Differences from Java interfaces
    - No static fields
    - All non-primitive parameters must be labeled by "direction"
    - Methods (& AIDL interfaces themselves) can be defined as **oneway**
      - **oneway** method invocations don't block the caller

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
  - Differences from Java interfaces
    - No static fields
    - All non-primitive parameters must be labeled by "direction"
    - Methods (& AIDL interfaces themselves) can be defined as **oneway**
    - Methods cannot throw exceptions

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

```
interface IDropBoxManagerService {
  void add(in DropBoxManager.Entry entry);
  ...
  DropBoxManager.Entry getNextEntry(String tag, long millis);
}
```

  - Similarities with Java interfaces
  - Differences from Java interfaces
    - No static fields
    - All non-primitive parameters must be labeled by "direction"
    - Methods (& AIDL interfaces themselves) can be defined as **oneway**
    - Methods cannot throw exceptions
    - Interfaces can't inherit from other interfaces

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax

- Supported Java primitives
  - boolean, boolean[], byte, byte[], char[], int, int[], long, long[], float, float[], double, double[]
  - java.lang.CharSequence, java.lang.String

```
interface IEmailService {
   …
   boolean createFolder(long accountId,
                        String name);
   boolean deleteFolder(long accountId,
                        String name);
   boolean renameFolder(long accountId,
                        String oldName,
                        String newName);
```

packages/apps/Email/emailcommon/src/com/android/emailcommon/service/IEmailService.aidl

AIDL Syntax & Supported Data Types

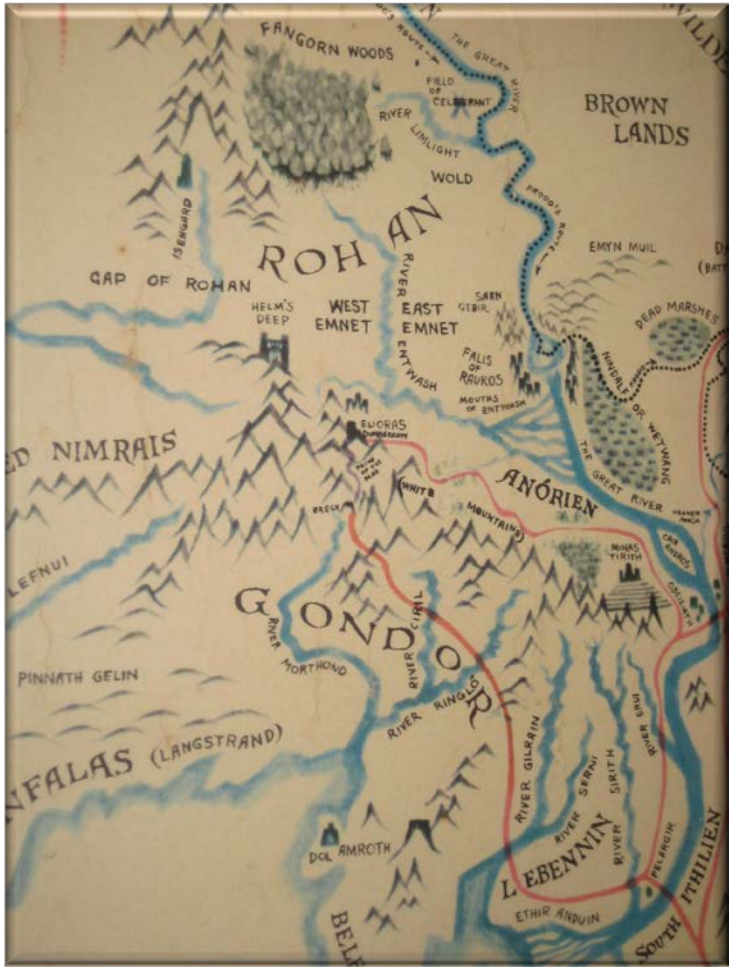
- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax
- Supported Java primitives
- java.util.List
  - Uses java.util.ArrayList internally
  - List elements must be valid AIDL data types
  - Generic lists supported

```
oneway interface
INetworkQueryServiceCallback {
  void onQueryComplete
    (in List<OperatorInfo>
        networkInfoArray,
     int status);
}
```

packages/apps/Phone/src/com/android/phone/INetworkQueryServiceCallback.aidl

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax
- Supported Java primitives
- java.util.List
- Java.util.Map
  - Uses java.util.HashMap internally
  - Map elements must be valid AIDL data types
  - Generic maps *not* supported
  - Not widely used (no use in Android)



AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax
- Supported Java primitives
- java.util.List
- Java.util.Map
- Classes implementing the Parcelable protocol

```
public class StatusBarIcon
             implements Parcelable {
  ...
  public void readFromParcel(Parcel in)
  { ... }
  public void writeToParcel
    (Parcel out, int flags) { ... }
}
```

Java source file

```
parcelable StatusBarIcon;

oneway interface IStatusBar {
  void setIcon(int index, in StatusBarIcon icon);
  ...
}
```

AIDL source file

developer.android.com/guide/components/aidl.html#PassingObjects has more

AIDL Syntax & Supported Data Types

- AIDL allows application developers to declare their "business" logic methods using a Java-like interface syntax
- Supported Java primitives
- java.util.List
- Java.util.Map
- Classes implementing the Parcelable protocol

```
public class StatusBarIcon
             implements Parcelable {
  ...
  public void readFromParcel(Parcel in)
  { ... }
  public void writeToParcel
    (Parcel out, int flags) { ... }
}
```

Java source file

```
parcelable StatusBarIcon;

oneway interface IStatusBar {
  void setIcon(int index, in StatusBarIcon icon);
  ...
}
```

AIDL source file

frameworks/base/core/java/com/android/internal/statusbar/IStatusBar.aidl

Summary

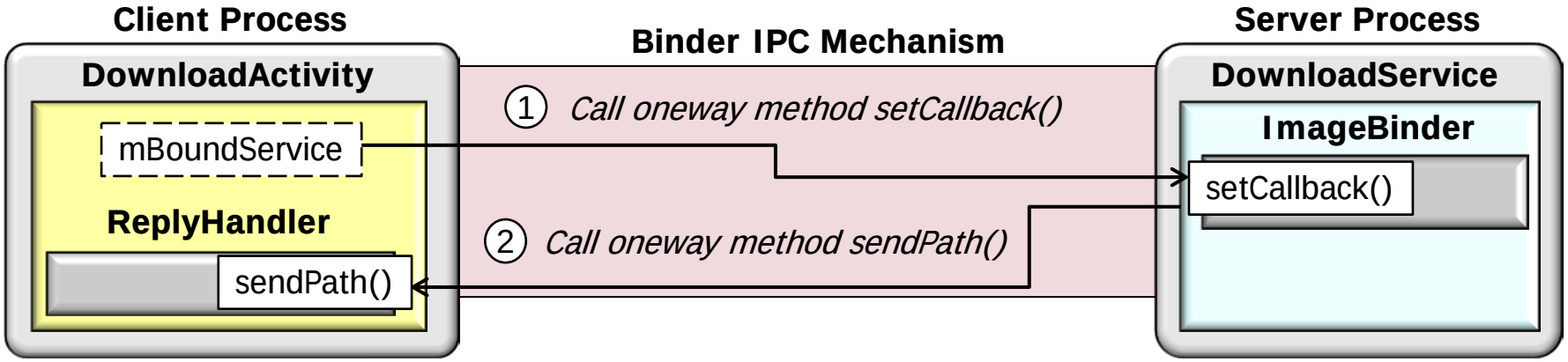
- AIDL uses a simple syntax that lets you declare an interface with one or more methods that can take parameters & return values

```
InterfaceDeclaration:
    interface Identifier InterfaceBody
InterfaceBody:
    { { InterfaceBodyDeclaration } }
InterfaceBodyDeclaration:
    ;
    InterfaceMethodDecl
InterfaceMethodDecl:
    Type Identifier InterfaceMethodDeclaratorRest
InterfaceMethodDeclaratorRest:
    FormalParameters
```

AIDL interfaces are a subset of Java interfaces

Summary

- AIDL uses a simple syntax that lets you declare an interface with one or more methods that can take parameters & return values
- The parameters & return values can be of any supported type, even other AIDL-generated interfaces
  - Interface methods that are passed parameters of other interfaces are commonly used to implement asynchronous one-way callbacks



Summary

- AIDL uses a simple syntax that lets you declare an interface with one or more methods that can take parameters & return values
- The parameters & return values can be of any supported type, even other AIDL-generated interfaces
- You must construct the .aidl file using a subset of the Java programming language
  - Each .aidl file must define a single interface & requires only the interface declaration & method signatures

```
interface Idownload {
    oneway void setCallback(in IDownloadCallback callback);
}
```

IDownload.aidl source file

IDownloadCallback.aidl source file

```
interface IDownloadCallback {
  oneway void sendPath(in String path);
}
```