Resource Forks in Genesis II

Mark Morgan

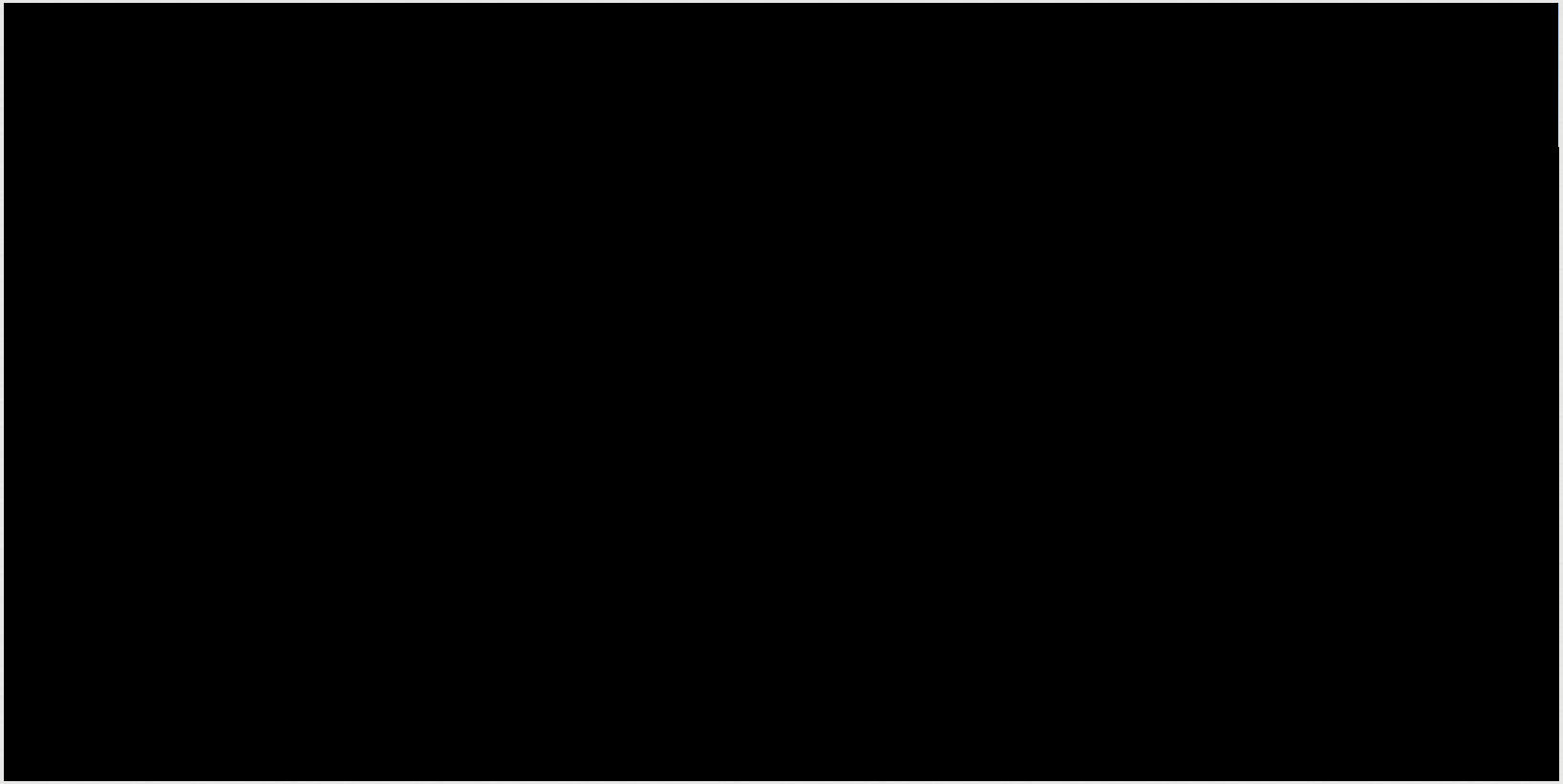
Outline

- What is a resource fork and why use them?
- Types of resource forks.
- Implementing resource forks.

What is a resource fork?

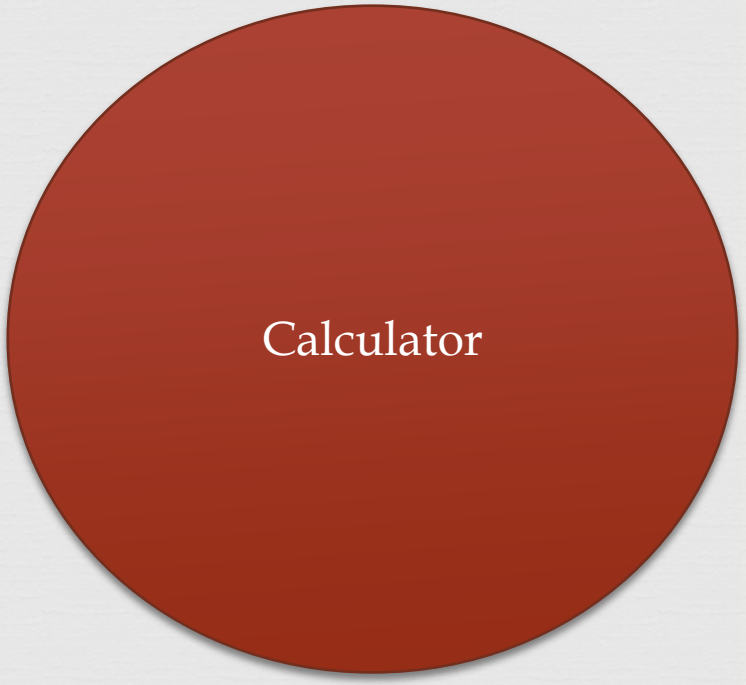
- A file system view of a resource
- A rooted directory structure
- The concept has been around a while
  - Plan 9 Operating System
  - Mac OS X
  - /proc on many *NIX OSs

A resource fork is a view



A resource fork is a directory structure

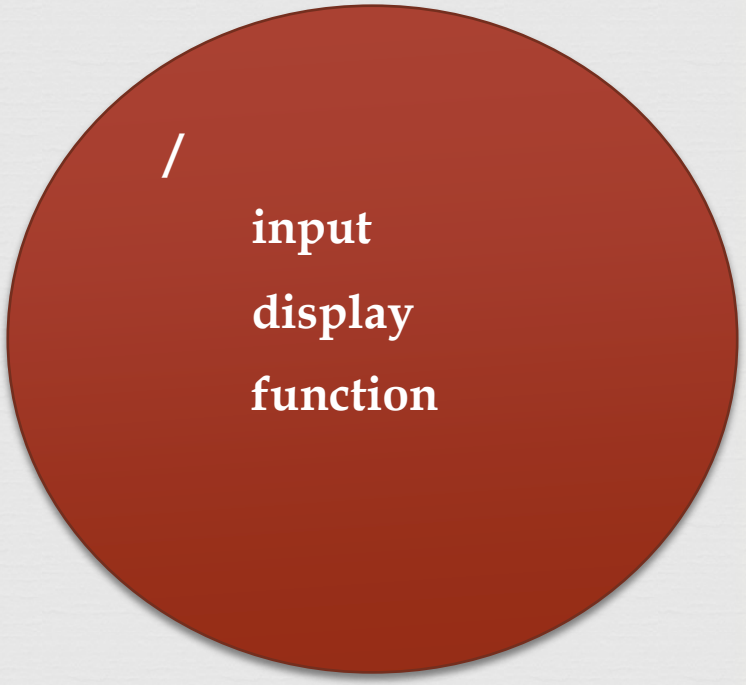
```
Calculator c;

c.inputNumber(7);
c.inputNumber(42);
c.add();
c.getNumber();
```



A resource fork is a directory structure

cd /Calculator

echo “7” > input
echo “42” > input
echo “add” > function
cat display



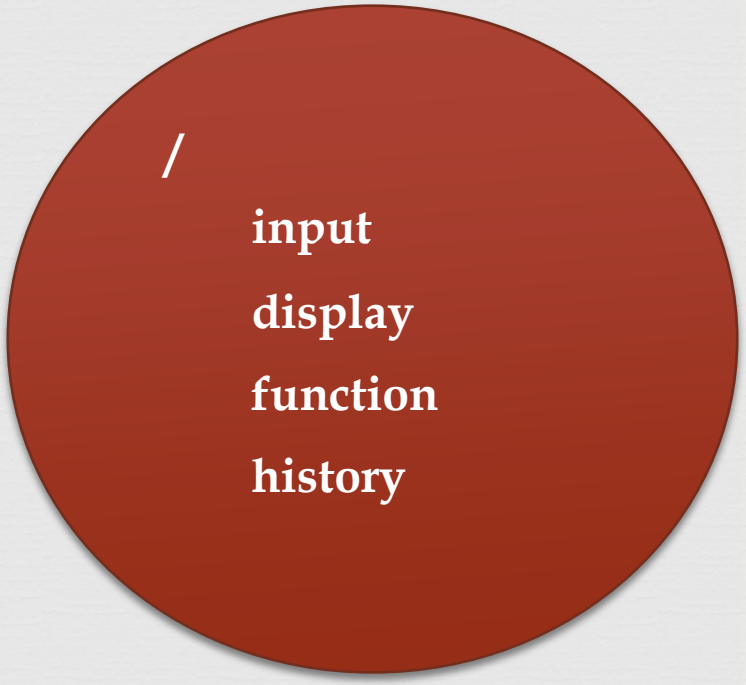
A resource fork is a directory structure

cd /Calculator

echo “7” > input
echo “42” > input
echo “add” > function
cat display

cat history

```
[21 March 2011: 10:00:00 AM] 7 input
[21 March 2011: 10:00:01 AM] 42 input
[21 March 2011: 10:00:02 AM] add requested
```



Outline

- What is a resource fork and why use them?
- Types of resource forks.
- Implementing resource forks.

Types of Resource Forks

- A resource fork is essentially a directory structure for use in file system views
- Technically, a resource fork is directory or a file
- In the grid file, since a directory can "contain" any grid resource, resource forks can contain any grid resource
- For Genesis II, a resource fork is any grid resource, or
  - An RNS
  - A RandomByteIO
  - A StreamableByteIO
  - A StreamableByteIOFactory

ResourceFork Interface

- **Attributes**
  - forkPath: String
- **Methods**
  - registerNotificationHandler(multiplexer: NotificationMultiplexer)
  - destroy()
  - describe()

RNSResourceFork extends ResourceFork

- **Attributes**
- **Methods**
  - list([entryName: String]): Iterable
  - add(entryName: String, entry: EPR)
  - remove(entryName: String): EPR
  - createFile(fileName: String): EPR
  - mkdir(dirName: String): EPR

ByteIOResourceFork extends ResourceFork

- **Attributes**
  - **size: long**
  - **createTime: Calendar**
  - **modificationTime: Calendar**
  - **accessTime: Calendar**
  - **readable: boolean**
  - **writable: boolean**
- **Methods**

RandomByteIOResourceFork extends ByteIOResourceFork

- **Attributes**
- **Methods**
  - read(offset: long, destination: ByteBuffer)
  - write(offset: long, source: ByteBuffer)
  - truncAppend(offset: newSize, source: ByteBuffer)

StreamableByteIOResourceFork extends ByteIOResourceFork

- **Attributes**
  - position: long
  - seekable: boolean
  - endOfStream: boolean
  - destroyOnClose: boolean
- **Methods**
  - seekRead(origin: SeekOrigin, destination: ByteBuffer)
  - seekWrite(origin: SeekOrigin, source: ByteBuffer)

StreamableByteIOFactoryResourceFork extends ByteIOResourceFork

- **Attributes**
- **Methods**
  - snapshotState(sink: OutputStream)
  - modifyState(source: InputStream)

Pre-defined Fork Types

- ReadOnlyRNSFork ← RNSResourceFork
- StaticRNSResourceFork ← RNSResourceFork
- SimpleStateResourceFork<Type> ← StreamableByteIOFactoryResourceFork

Read-Only RNS Fork

- Simply implements the RNS fork interface for all "write" methods so that they throw an exception

Static RNS Resource Fork

- A version of an RNS resource fork that has a "set-up" method where you add a list of name/resource-fork pairs that never change (the name/fork pairs don't change – the forks themselves can change as they want).
- This is one of the more common "root" resource forks.

Simple State Resource Fork

- A resource fork that is a version of the StreamableByteIOFactoryResourceFork
- This fork takes a type which it "serializes" for snapshots and which it reads from a "serialized" state for modifications
- Default serialization/deserialization is text-based, but you can override with your own.
- Serialization is really more of a "parse" than a de-serialize

Resource Forks and Security

- Currently, all entries for a rooted resource fork tree share the same ACLs
  - There are some papers describing something I have called “Inheritable ACLs” that would allow this to change
- You set the RWX (read-write-execute) category for methods using the same annotation (@RWXMapping and @RWXMappingResolver) that normal services use

Resource Forks and EPRs

<Endpoint>
<Address>https://titus/axis/services/QueuePortType</Address>
<ReferenceParameters>
<resource-key>EECA18B6-DF8F-64DE-EBAF-46FD45F005F3</resource-key>
<resource-fork>
rO0ABXNyAEZlZHUudmlyZ2luaWEudmNnci5nZW5paS5jb250YWluZXIucmZvcmsuRGVmYXVsdFJlc291cmNlRm9ya0luZm9ybWF0aW9uAAAAAAAAAAADAANMAAxfY29uc3RydWN0b3J0AB9MamF2YS9sYW5nL3JlZmxlY3QvQ29uc3RydWN0b3I7TAAJX2ZvcmtQYXRodAASTGphdmEvbGFuZy9TdHJpbmc7TAASX3Jlc291cmNlRm9ya0NsYXNzdAARTGphdmEvbGFuZy9DbGFzczt4cHdkAERlZHUudmlyZ2luaWEudmNnci5nZW5paS5jb250YWluZXIucTIuZm9ya3MuUmVzb3VyY2VNYW5hZ2VtZW50Q21kRm9yawAcL3Jlc291cmNlLW1hbmFnZW1lbnQvc3lzdGVtWHg=
</resource-fork>
</ReferenceParameters>
<Metadata>
<EndpointIdentifier>urn:ws-naming:epi:E33FA255-DB52-C71F-A847-3C828939E0F5:fork-path:/resource-management/systemX</EndpointIdentifier>
</Metadata>
</Endpoint>

Outline

- What is a resource fork and why use them?
- Types of resource forks.
- Implementing resource forks.

Implementing a Resource Fork

- Step 1: Implementing a Resource
  - Your service has to implement the ResourceForkPortType
  - Derive your service off of ResourceForkBase, NOT GenesisIIBase
- Step 2: Create a Root RNS Resource Fork
- Step 3: Annotate your service with the **@ForkRoot** annotation
- Start adding resource forks to your hierarchy either statically, or programmatically

Step 1: Implementing the Resource

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="Queue"
    xmlns="http://vcgr.cs.virginia.edu/genii/queue" ...>

    <wsdl:import
        namespace="http://vcgr.cs.virginia.edu/genii/resource-
fork"
        location="./rfork-generated.wsdl"/>

    <wsdl:portType name="QueuePortType">
        <genii-ext:extend portType="rfork:ResourceForkPortType"/>


</wsdl:definitions>
```

Step 1: Implementing the Resource Cont.

```
@ForkRoot(RootRNSFork.class)
@GeniiServiceConfiguration(
    resourceProvider=QueueDBResourceProvider.class)
@GridDependency(GeniiBESServiceImpl.class)
public class QueueServiceImpl extends ResourceForkBaseService
    implements QueuePortType
{
        ...
}
```

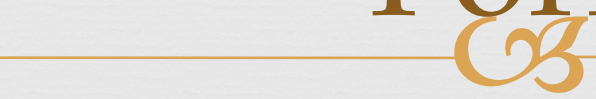
Step 2: Create a Root RNS Fork

```
public class RootRNSFork extends StaticRNSResourceFork
{
        public RootRNSFork(ResourceForkService service, String forkPath)
        {
                super(service, forkPath);
        }

        @Override
        protected void addEntries(Map<String, ResourceForkInformation> entries)
        {
                addDefaultEntry("resources", ResourcesRNSFork.class);
                addDefaultEntry("resource-management", ResourceManagementRNSFork.class);
                addDefaultEntry("jobs", JobsRNSFork.class);
                addDefaultEntry("submission-point", JobSubmissionFork.class);
                addDefaultEntry("summary", QueueSummaryResourceFork.class);
                addDefaultEntry("is-scheduling-jobs", IsSchedulingPropertyFork.class);
        }
}
```

Step 3: Annotate Your Service with @ForkRoot

```
@ForkRoot(RootRNSFork.class)
@GeniiServiceConfiguration(
    resourceProvider=QueueDBResourceProvider.class)
@GridDependency(GeniiBESServiceImpl.class)
public class QueueServiceImpl extends ResourceForkBaseService
    implements QueuePortType
{
        ...
}
```

Using a Vanilla RNS Fork



See Queue's ResourcesRNSFork

Using a Vanilla RNS Fork (with sub-forks)

See Queue's JobListingRNSFork

Using a Stream Factory Resource Fork

See Queue's JobInformationFork

Using a SimpleState Resource Fork

See Queue's IsSchedulingPropertyFork

Using a Random ByteIO Resource Fork

See LightWeightExport's LightWeightExportFileFork