EOF-Using EOF-Custom EOAdaptor

It's surprisingly easy to create a simple custom EOAdaptor. You can look at JavaFSAdaptor, which is part of Project Wonder, and you can also look at JavaMemoryAdaptor, also in Wonder.

Here is a little example on how the create an EOAdaptor:

Creating an EOAdaptor can be very straightforward (or very complicated) depending on the underlying data source. In this example, and for the sake of keeping thing simple, I'm using the file system as a data source (well, at least, how java.io.File sees it).
At a bare minimum you need to implement three classes: EOAdaptor, EOAdaptorContext, EOAdaptorChannel.

[EOAdaptor JavaDoc]

Not much in here. The EOAdaptor is mostly a rendezvous point to access other functionality in your adaptor. You need to implement a couple of abstract methods though:

- assertConnectionDictionaryIsValid() to check the validity of a connection dictionary.
- createAdaptorContext() to create your own EOAdaptorContext instance.
- defaultExpressionClass() to return a relevant EOSQLExpression class.
- isValidQualifierType() to decide if a qualifier can be used in a SQL where clause.
- synchronizationFactory() to return a EOSchemaGeneration helper.

In the case of a file system there is not much to do:

```
public void assertConnectionDictionaryIsValid()
 {
 }

 public EOAdaptorContext createAdaptorContext()
 {
  return new FSAdaptorContext( this );
 }

 public Class defaultExpressionClass()
 {
  throw new UnsupportedOperationException
( "FSAdaptor.defaultExpressionClass" );
 }

 public EOSQLExpressionFactory expressionFactory()
 {
  throw new UnsupportedOperationException
( "FSAdaptor.expressionFactory" );
 }

 public boolean isValidQualifierType(String aTypeName, EOModel aModel)
 {
  return true;
 }

 public EOSchemaGeneration synchronizationFactory()
 {
  throw new UnsupportedOperationException
( "FSAdaptor.synchronizationFactory" );
 }
```

The only important method is createAdaptorContext().

[EOAdaptorContext JavaDoc]

The adaptor context handles "transactions". It's usually pretty simple if your data store support that concept. Not much to do here for a file system:

```
public void beginTransaction()
 {
  if ( _hasTransaction == false )
  {
   _hasTransaction = true;

   this.transactionDidBegin();
  }
 }

 public void commitTransaction()
 {
  if ( _hasTransaction == true )
  {
   _hasTransaction = false;

   this.transactionDidCommit();
  }
 }

 public EOAdaptorChannel createAdaptorChannel()
 {
  return new FSAdaptorChannel( this );
 }

 public void handleDroppedConnection()
 {
 }

 public void rollbackTransaction()
 {
  throw new UnsupportedOperationException
( "FSAdaptorContext.rollbackTransaction" );
 }
```

Again createAdaptorChannel() is the only method that you really care about in this case.

[EOAdaptorChannel JavaDoc]

That's the real thing. It's where most of the work is done. It's important to get this one right. Depending on your back end, it's simply an exercise in translating EOQualifiers into something that your data storage will understand. Here a the most important methods:

- deleteRowsDescribedByQualifier() to handle deletion of "row" or something equivalent.
- insertRow() to create a "row".
- updateValuesInRowsDescribedByQualifier() to update a "row".

Finally, fetching "rows" is a two steps process: first you need to setup the "context" of what you are going to fetch and then fetch it:

- selectAttributes() is where you set your fetch "context".
- fetchRow is the actual fetching one row at the time.

In the case of a file system here is what you could have:

- deleteRowsDescribedByQualifier will delete some files (aka "rows" based) on a qualifier.

```
public int deleteRowsDescribedByQualifier(EOQualifier aQualifier,
EOEntity anEntity)
 {
  if ( aQualifier != null )
  {
   if ( anEntity != null )
   {
    NSArray someFiles =
FSQualifierHandler.filesWithQualifier( aQualifier );

    if ( someFiles != null )
    {
     someFiles = this.filteredArrayWithEntity( someFiles, anEntity );

     if ( someFiles != null )
     {
      int count = someFiles.count();
      int counter = 0;

      for ( int index = 0; index < count; index++ )
      {
       File aFile = (File) someFiles.objectAtIndex( index );

       if ( aFile.delete() == true )
       {
        counter += 1;
       }
      }

      return counter;
     }
    }

    return 0;
   }

   throw new IllegalArgumentException
( "FSAdaptorChannel.deleteRowsDescribedByQualifier: null entity." );
  }

  throw new IllegalArgumentException
( "FSAdaptorChannel.deleteRowsDescribedByQualifier: null qualifier."
);
 }
```

- insertRow will create a file or directory.

```
public void insertRow(NSDictionary aRow, EOEntity anEntity)
 {
  if ( aRow != null )
  {
   if ( anEntity != null )
   {
    String aPath = (String) aRow.objectForKey( "absolutePath" );

    if ( aPath != null )
    {
     File aFile = new File( aPath );

     try
     {
      if ( anEntity.name().equals( "FSDirectory" ) == true )
      {
       aFile.mkdirs();
      }
      else
      {
       aFile.createNewFile();
      }
     }
     catch(Exception anException)
     {
      throw new RuntimeException
( "FSAdaptorChannel.insertRow: " + anException );
     }

     return;
    }

    throw new IllegalArgumentException
( "FSAdaptorChannel.insertRow: null absolutePath." );
   }

   throw new IllegalArgumentException
( "FSAdaptorChannel.insertRow: null entity." );
  }

  throw new IllegalArgumentException
( "FSAdaptorChannel.insertRow: null row." );
 }
```

- updateValuesInRowsDescribedByQualifier will change a file attributes.

```
public int updateValuesInRowsDescribedByQualifier(NSDictionary
aRow, EOQualifier aQualifier, EOEntity anEntity)
 {
```

```
  if ( aRow != null )
  {
   if ( aQualifier != null )
   {
    if ( anEntity != null )
    {
     NSArray someFiles =
FSQualifierHandler.filesWithQualifier( aQualifier );

     if ( someFiles != null )
     {
      someFiles = this.filteredArrayWithEntity( someFiles, anEntity );

      if ( someFiles != null )
      {
       int count = someFiles.count();

       for ( int index = 0; index < count; index++ )
       {
        File aFile = (File) someFiles.objectAtIndex( index );
        NSArray someKeys = aRow.allKeys();
        int keyCount = someKeys.count();

        for ( int keyIndex = 0; keyIndex < count; keyIndex++ )
        {
         Object  aKey = someKeys.objectAtIndex( keyIndex );
         EOAttribute anAttribute =
anEntity.attributeNamed( aKey.toString() );

         if ( anAttribute != null )
         {
          Object aValue = aRow.objectForKey( aKey );


NSKeyValueCoding.DefaultImplementation.takeValueForKey( aFile, aValue,
anAttribute.columnName() );
         }
        }

       }

       return count;
      }
     }

     return 0;
    }

    throw new IllegalArgumentException
( "FSAdaptorChannel.updateValuesInRowsDescribedByQualifier: null
entity." );
   }
```

```
   throw new IllegalArgumentException
( "FSAdaptorChannel.updateValuesInRowsDescribedByQualifier: null
qualifier." );
  }

  throw new IllegalArgumentException
( "FSAdaptorChannel.updateValuesInRowsDescribedByQualifier: null row."
```

```
  );
   }
```

- selectAttributes will retrieve some files based on a qualifier.

```
public void selectAttributes(NSArray someAttributes,
EOFetchSpecification aFetchSpecification, boolean shouldLock, EOEntity
anEntity)
 {
  if ( someAttributes != null )
  {
   this.setAttributesToFetch( someAttributes );

   if ( aFetchSpecification != null )
   {
    if ( anEntity != null )
    {
     NSArray someFiles =
FSQualifierHandler.filesWithQualifier( aFetchSpecification.qualifier()
);

     if ( someFiles != null )
     {
      NSArray someSortOrderings = aFetchSpecification.sortOrderings();

      if ( someSortOrderings != null )
      {
       someFiles =
EOSortOrdering.sortedArrayUsingKeyOrderArray( someFiles,
someSortOrderings );
      }

      someFiles = this.filteredArrayWithEntity( someFiles, anEntity );

      if ( someFiles != null )
      {
       this.files().addObjectsFromArray( someFiles );
      }
     }

     return;
    }

    throw new IllegalArgumentException
( "FSAdaptorChannel.selectAttributes: null entity." );
   }

   throw new IllegalArgumentException
( "FSAdaptorChannel.selectAttributes: null fetch specification." );
  }

  throw new IllegalArgumentException
( "FSAdaptorChannel.selectAttributes: null attributes." );
 }
```

- fetchRow will "fetch" a dictionary representation of a file.

```
public NSMutableDictionary fetchRow()
 {
  File aFile = (File) this.files().lastObject();

  if ( aFile != null )
  {
   this.files().removeLastObject();

   return this.dictionaryForFileWithAttributes( aFile,
this.attributesToFetch() );
  }

  return null;
 }
```

That could be it. Nothing to involved. However there is two extra methods that make sense to implement: describeTableNames and describeModelWithTableNames.

- describeTableNames() return an array of "table" (or something conceptually equivalent) existing in you data back end.
- describeModelWithTableNames() will create a default model for some of these "tables".

This is very useful to get you started in EOModeler instead of having to create the entire model by hand. In the case of a file system, there is only one model:

```
public EOModel describeModelWithTableNames(NSArray anArray)
 {
  return new EOModel( this.defaultModelPath() );
 }
```

And voilà. No you have a toy file system adaptor (aka JavaFSAdaptor).

- To "fetch" some files, simply create a qualifier with some path attributes (eg absolutePath):

```
EOQualifier  aPathQualifier = new EOKeyValueQualifier
( "absolutePath", EOQualifier.QualifierOperatorEqual,
System.getProperty
( "user.home" ) );
```

- To get all the files in a directory, use the "parent" attribute:

```
EOQualifier  aPathQualifier = new EOKeyValueQualifier( "parent",
EOQualifier.QualifierOperatorEqual, System.getProperty( "user.home" )
);
```

Once you have your qualifier, simply build a fetch specification for it and get your "file" EOs:

```
EOFetchSpecification aFetchSpecification = new EOFetchSpecification
( "FSDirectory", aQualifier, null );
NSArray    someObjects =
anEditingContext.objectsWithFetchSpecification( aFetchSpecification );
```

You can use FSFile to retrieve files only. FSDirectory for, er, directories. And FSItem for both. FSDirectory have two handy relationships: "files" and "directories". Check the attached model.

- To "create" a file (or directory), you can do something along those lines:

```
EOClassDescription aClassDescription =
EOClassDescription.classDescriptionForEntityName( "FSDirectory" );
EOEnterpriseObject anObject =
aClassDescription.createInstanceWithEditingContext( anEditingContext,
null );

anObject.takeValueForKey
( "/Users/rszwarc/tmp/JavaFSAdaptor/TestFSDirectory", "absolutePath"
);

anEditingContext.insertObject( anObject );

anEditingContext.saveChanges();
```

- Finally, do delete a file:

```
anEditingContext.deleteObject( anObject );

anEditingContext.saveChanges();
```