Hierarchy

FAQ

For Hierarchy Beta-Version

Created by *Unconventional Thinking*

Last update: June 15th, 2014

Introduction

This FAQ contains solutions to common problems you'd run across using the Hierarchy metacompiler. It's organized into different sections. You can jump to the section that most relates to the problem you're having by clicking on one the the links below.

System requirements

- Make sure you have Java 6 or above installed on your system
- *NOTE: Java 8 is supported, but as of yet, not it's new features like Lamdas!! Stick to Java 7 syntax!*
- Make sure you have Ant 1.6 or above installed on your system and can run it from the command line.
- If you'd like to use the sample web project, install Tomcat 6.0 or above on your system
- The Hierarchy.jar should be in your projects classpath. **And, it's dependent jars (lib/sablecc.jar, lib/commons-collections-#.#.#.jar) should be in subfolder of the Hierarchy.jar directory, called lib!**

Current Know Bugs and Limitations of the Metacompiler – IMPORTANT!:

- **Java 8 –** If you're using Java 8 as your compiler:
  - *Stick with Java 7 syntax!* Java 8's new features (like Lamdas) is not supported!

- **In Embedded Java files (.mjava)**
  - Inner classes are currently not supported!
  - Having extra classes in one .mjava file are not supported – for instance, having one public class followed by one with package-level access.
- You can't put any matrix files (.mjava, .matrix, and .schema) in the default package. They must have a package!

- When creating schemas, there's a bug with field names. Currently, you can't start a fieldname with underscore: ex +%:_MyField

- If you create a jar file that as an embedded Java class. And, this embedded Java class has a **public class variable** that is a "descriptor var with access-type", and, *your current project's code* tries to access that desc var, the access-type info is currently lost and not be used!!
- Also, similarly, if you create a jar file that has an embedded Java class file. And, in this embedded Java class, you have "Descriptor Variables with access types" as **parameters or return types to a method,** and, *your current project's code* tries to access that desc var, the access-type info is currently lost and not be used!!
- If you have any problems with using *regular* Java files (you know, with .java extension) with .mjava files, refer to the section on "Dealing with Java Dependencies" in Chapter 7 in the Developer Guide.

Project Configuration Problems for Different IDE's

Eclipse – these are problems you might find specific to working with Hierarchy in Eclipse

- You may have to fix your project's JRE settings, especially for the web projects. In your "Markers" window pane, if you see the error:

```
Java Build Path Problems
     x Unbound classpath container: ' JRE System Library [jdk1.6.0_21]
     in project …
```

  This means your JRE settings for some of your sample projects are incorrect. To fix the JRE problem:

  a. Open your project's Properties dialog.
  b. In the left nav, select "Java Build Path."
  c. Click on the "Libraries" tab.

d. You should see a red ‘x’ by the library, “JRE System Library.” Select this entry and then click the Edit button.
e. A dialog will pop up giving you three options for selecting a JRE. It doesn’t matter which one you chose, as long as the JRE version is 6, and that it resolves correctly!
f. Check that this worked. In the Markers pane, the error should have gone away for this project – note that you may have more than one sample project with this same error, so even if this error still shows up after the fix, double check to make sure that the one for the project you just fixed has gone away.

Matrix Files (.matrix)

- **If you have problems using a schema, don’t forget you need to import it in:**
    - Refer to in the section “Using a Schema in a Matrix: Importing the Schema,” in Chapter 4 – “Schemas” in the Developer Guide

- If you have any problems with using *regular* Java files (you know, with .java extension) with .matrix files, refer to the section on “Dealing with Java Dependencies” in Chapter 7 in the Developer Guide.

- If you get a syntax error in the name you used for the matrix:
    - Make sure you don’t use any reserved words in the matrix name! For example:

```
MATRIX CARINFO.MATRIX { …  // bad!! - the “.MATRIX” is a reserved word!
```

Schema Files (.schema)

Problems with defining the types of your fields:

- If your metacompiler throws an error on your schema that says something like this:

```
MetaCompiling Schema Code:
Output is:
done! 1 schema files compiled.

Generate FieldSets:
First, Load the appControl to create the schemas
Output is:
maintests\samples\FieldSetTuple__maintests$__$samples$_CC_$WEB$__$FORM$_S_$F
ORM$__$REQUIRED.java:30: cannot find symbol
symbol  : class MyType
location: class
maintests.samples.FieldSetTuple__maintests$__$samples$_CC_$WEB$__$FORM$_S_$F
ORM$__$REQUIRED MyType IsRequired;
```

Then, you may not have defined one of the types in your fields correctly.

Solution:

- **If you use a primitive type as the type of one of the fields in your schema, use the "symbol with quotes" syntax to specify it** – the reason is because primitive types are Java keywords, and can't be used as symbol names directly**.**

```
  FIELD.TYPES:  { :String, :String, :"int"}; // NOTE: String is an
                                   // object type and is not a Java keyword
```

For more information, refer to two sections in Chapter 3 of the Developer Guide:
  - **"Summary of How to Use Symbols"**
  - **"Summary of When to Use a Specific Type of Symbol in a Schema"**

- **If you use a Java Object as the type of one of the fields in your schema, use the "symbol with quotes" syntax to specify it. And, put the *full* type name in a *single* set of quotes!**

```
  FIELD.TYPES:  { +:"java.util.Date", :String, :"int"};
```

* And, remember, don't forget to add a '+' before the symbol name for object types! The reason is because almost all type definitions are symbols that haven't been created before, so you need the '+' to create them (oh, and the reason you don't need a plus for the primitive types like `:"int"` is because Hierarchy pre-creates these commonly used symbols).

For more information, refer to two sections in Chapter 3 of the Developer Guide:
  - **"Summary of How to Use Symbols"**
  - **"Summary of When to Use a Specific Type of Symbol in a Schema"**

- If you get a syntax error in the name you used for the schema, or in the descriptor defintions:
    - Make sure you don't use any reserved words in the schema or descritpor name! For example:

```
SCHEMA CARINFO.SET { …  // bad!! "SET" is a reserved word!
   DESCRIPTOR +:%CAR.INTERIOR.ITEM { // bad!! "ITEM" is reserved!
   DESCRIPTOR +:%CAR.EXTERIORITEM { // Good! ITEM is apart of EXTERIOR,
                                    // which is okay.
```

Limitations on Using Schemas

- You can't put an mjava in the default package. It must have a package!

Other Problems you might Have with Schemas

- If you have any problems with using *regular* Java files (you know, with .java extension) with .schema files, refer to the section on "Dealing with Java Dependencies" in Chapter 7 in the Developer Guide.

Embedded Java Files (.mjava)

- **For all methods in your class that use matrix accesses, the ANNOTATIONS section is required –** In the generate Java code, if you get an error dealing with annotations, make sure that all your methods that use a matrix access include an annotation at the end. And, add in a DEFAULT annotation handler to it.

```
ANNOTATIONS {
    DEFAULT: {
      return null;
    }
}
```

  NOTE: The metacompiler helps you by letting you know if you forget to add an ANNOTATION handler to a method with Matrix accesses.

- **Make sure you're not mixing up field access, '**`:>`', **with descriptor access, '**`->`'

- **If you get this error during metacompilation, then you probably need to import a matrix that your using:**

```
com\mypackage\MyClass__MatrixWorker.java:29: cannot find symbol
symbol  : variable MyMatrix
location: class com.fictitiouscorp.app.config.ConfigProcessor__MatrixWorker
                if (DescriptorUtilities.validDescriptors(executeInfo,
MyMatrix.Matrix)) {
```

  - For more information, in the Developer Guide, refer to the section: "Using a Matrix in an Embedded Java File: Importing the Matrix," in "Chapter 6 – Accessing Matrices in your Embedded Java Code".

- **Limitations**
  - Inner classes are currently not supported!
  - Having extra, non-public classes in an embedded file is not supported!
  - You can't put an mjava in the default package. It must have a package!

Metacompiling the Generated Java Code

- If you ever have problems metacompiling the code, refer to chapter 7 in the Developer Guide, on ***"Using the Metacompiler: How to metacompile and debug your code."*** It has a ton of information about the metacompiler and ***is the first place to look for info on how to solve metacompilation problems!***

- **Problems related to: Placing generated Java files back into the project's main src directory:** More specifically, we are talking about projects that are setup so that your metacompiler metacompiles Java files and places them back into the project's main, Java directory (instead of placing them in a separate, build directory).

  - Problem: Sometimes after metacompilation, the IDE doesn't see all these newly generated files that have been placed back into the src directory.
    - Solution: We've found that it tends to miss the AppSymbols.java file. If you find you can't compile the generated java files, sometimes you have to make a small change to AppSymbols.java file (like add a random space character) and re-save it for the IDE to pick up the file.
  - Problem: Let's say you've already run the metacompiler once, and it's metacompiled correctly and placed the generated Java files back into your src directory. This initial compilation works, *but* when you try to rerun the metacompiler with all the *exact same metacode*, the metacompiler spits out a "null reference" exception, especially while trying to compile the Embedded Java files. *And note that we mentioned that the there were no changes to the metacode from the first metacompilation to the second, so it should have worked!*
    - What's happening is in this second run, the metacompiler is reading in the generated Java class and causing some strange situation to internally occur with language symbols.
    - Solution: Clean out the generated metacode before each run. You can do this automatically by going your "metacompilation.properties" file and turn on the clean setting:

```
clean=true
```

- **For Eclipse users:**

  - **after you've generated .java files using the meta-compiler, for these generated .java files to show up in your Eclipse project, you need to "refresh" the project.** You can do this in the package explorer window by right clicking on the project and selecting **"refresh"** from the pop up menu. In Eclipse, you should now to able to see the generated .java files in the src directory.
    - **After you refresh, you need to build the project.**