Custom Code Overview

Your Game Code - Adding Custom Elements · https://namazustudios.com/docs/custom-code/element-structure/

Deploy custom code to your instance of Namazu Elements.

Elements is highly extensible and allows you to develop custom code. Elements 3 is written in Java21 and therefore any VM compatible languages will work seamlessly in Elements. Building your custom server side logic in Java-Based Languages has several distinct advantages.

- Java has one of the most complete ecosystems of readily available Open Source libraries available. Chances are, whatever you're looking for is available in a package on Maven Central.
- Java offers incredible server side performance through JIT Compliation, which essentially compiles your code to machine code as it is loaded.
- As of October 2023, it is estimated that the developer community size is approximately 17.1 Million developers, ensuring you will have no problems building your team long-term.
- An existing suite of powerful development tools including IDEs and profilers/benchmarking tools are readily available.

Check out our example Element project here!

Here's where you learn to develop your first Element.

Starting with Elements 3, we are now providing a way to develop custom extensions in Java Virtual Machine (or JVM) based languages. Refer to the Wikipedia page for a complete list and determine the best language which works for your project. The most popular choices are Java and Kotlin. Elements is natively written in Java which is 100% compatible with any other JVM based language.

We call our fundamental plug-in an Element. Each Element runs in an isolated environment using a ClassLoader which isolates Element from the rest of the system. A common problem Java developers face is conflicting dependencies within a single application, colloquially known as "Jar Hell."

Our approach allows you to use develop in an environment without having to worry about conflicts between the Elements 3 code base and your Element's code. We provide a set of interfaces, but isolate their implementations from your code. When developing an Element, these are the key points you must know:

- You can incorporate almost any existing Java code base into Elements that are based on standard Java frameworks. Elements 3 Currently Supports:
- Jakarta RS 4.0.0
- Jakarta Websockets 2.1.0
- You can include almost any third-party library into your code base without having to worry about it conflicting with the frameworks used to build Elements
- Note: You will need to copy all dependencies into your Element, except those as provided by Elements 3.
- An Element does not have strict isolation, nor does it run in a sandbox. This is by design, as trying to enforce strong encapsulation using a Security Manager (or similar) would introduce a lot of overhead for no benefits. Take special care when accessing the System ClassLoader.

Defining an Element

An Element exists as a Java package and, optionally, include all sub-packages recursively. Elements are similar to JPMS Modules but without the one jar per module restrictions.

The following example shows how to define an Element in your Java code. Note we do require that you define a `package-info` for the package. The following is the simplest Element possible.

```
// Defines the Package. Specifies as recursive.
@ElementDefinition(recursive = true)
package com.mystudio.mygame.api;

// Imports the Element SDK Annotation
import dev.getelements.elements.sdk.annotation.Element
```

This indicates that the package `com.mystudio.mygame.api` and all sub packages will be included in the Element for associated services.

Packaging an Element

Packaging an Element or collection of Elements uses a simple directory structure. Elements 3.0 will search for the following files and directories when loading the assets associated with an Element:

- `dev.getelements.element.attributes.properties` - A standard Java Properties file which define the Element's application
- `libs` - A directory containing jar files. The loader scans the directory for jar files and adds them to the Element's classpath.
- `classpath` - A directory containing assets to add directly to the Element's classpath.

Additionally, the following rules apply when scanning a directory for Elements.

- Any file not otherwise specified would be ignored.
- Any directory will be treated as a new Element, provided that that at least `libs` or `classpath` exist.
- **Note:** Elements defined in directories will inherit the classpath of the parent which allows you to put common code in the parent directory.

Next steps:

See our intro to Jakarta and Guice if you're not familiar with those tools, then check out Structuring an Element to get started!