SDK Overview

We have restructured the SDK to let you create modules with minimal startup time. This new format reduces configuration, eases dependency management and makes it easier to set up the development environment of your choosing.

In past versions of the SDK, developers were provided a sizable zip file, complete with jars, *javadocs*, this guide, and some third party dependencies. Builds were run using an Ant script, through the included *Build* directory. This all made sense when the original SDK was conceived – Ant imperfect but stable, Maven just stabilizing and Gradle wasn't even born. As technology evolves, we believe we need to evolve with it.

With repositories like Maven Central, and dependency management through Maven and Ivy, Java has moved to the cloud. With the new Ignition 7.8 SDK, we have taken the same approach to provide you with simple dependency management, modern build tools, and open source example modules on our Git hub page.

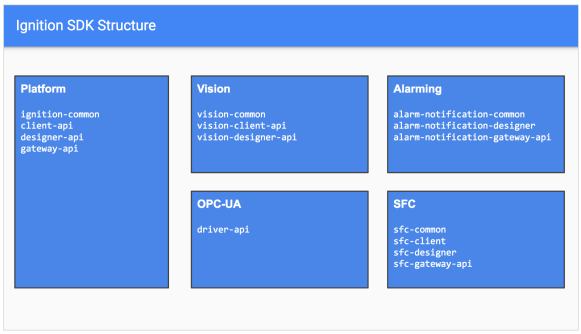
Fortunately, that does not mean we are abandoning existing modules. Existing modules which use the older Ant + Jar structure will continue to work as they were built. In addition, there are number of ways Ant can continue as your build process while also automatically resolving API dependencies from our Maven repository to easily support future versions of ignition. Libraries such as Mercury provide the ability to pull Maven dependencies into Ant tasks, or you can simply use ant to call Maven directly from Ant using a task .

Once those dependencies are resolved, your local maven cache will have them available for your build, avoiding the need for ongoing internet connection (unless/until you add more dependencies).

SDK Structure

The Ignition SDK consists of a collection of APIs that are provided as hosted *Artifacts* by the Inductive Automation Maven repository. If you are familiar with Maven Central, then using the Ignition SDK will feel very familiar.

The interfaces and resources that define the actual API, outlined below, are resolved by your build when it's time to compile your code into a module. If you are using a modern IDE or Text Editor with "Auto Import" abilities, your dependencies will be available just as if you had the jars locally. Note that the initial depedency resolution from the Inductive Automation Nexus repo will require an internet connection to download the Artifacts.



Resolve dependencies and build Ignition Modules in Ant using Maven

```
<target name="buildModl"
description="Calls Maven
package goal to build .
modl file">
    <exec dir="${source.
dir}\${projectName}"
executable="cmd">
        <arg value="/c"/>
        <arg value="${env.
MAVEN_HOME}\bin\mvn.bat"/>
        <arg line="clean
package" />
    </exec>
</target>
```

Javadocs

The JavaDoc technical documentation are compiled off of the API. They can be downloaded to your IDE or built off the maven repos. If you'd like to build the Javadocs, follow our "Getting Started" guide to create a module project, and then run the following Maven commands to get the docs built.

```
mvn dependency:resolve -Dclassifier=javadoc
```

API JARs

The API consists of the following JAR files that will need to be referenced as dependencies by your projects.

Core API and classes exposed by the Ignition platform

client-api.jar Needed for "client" scope
designer-api.jar Needed for "designer" scope
gateway-api.jar Needed for "gateway" scope

Software Design Tip

Maximize your code reuse, modularity and maintainability by using *common* directories for source that is used across multiple scopes/apis.

Vision APIs exposed by the Vision Module

- vis-client-api.jar
- vis-designer-api.jar
- vis-common.jar
    - *Common* files for both the client and designer.

Drivers APIs exposed by the OPC-UA module

- driver-api.jar
    - Used to create new drivers for the OPC-UA module.

Alarming APIs exposed by the Alarm-Notification