Overview of CREOSON Source Code

HTTP Server layer - creoson-server project

Jar Files: CreosonServer.jar, creoson-javadoc.jar

HTTP server

The main entry point for the server is **MainServer**. It establishes the HttpServer process which listens for requests and passes them off to the appropriate handlers.

HTTP Handlers

There are various handler classes for handling server requests. **JshellHttpHandler** passes off API requests to the JSON handler layer, and is the most important one. **ServerHttpHandler** handles special server-only requests and passes them off to the JSON handler in the same jar.

Website

The **/web** directory contains the website that is bundled with the HTTP Server. This includes the JSON files which define the API documentation (see "Other").

GenTemplates is a utility class that generates the JSON files for the API documentation in the web root.

Buildfiles

- **build.xml** will build CreosonServer.jar
- **build-javadoc.xml** will build creoson-javadoc.jar
- **build-zip.xml** will bundle all the jars into a the final Release ZIP files. This will also bundle in the **CreosonSetup** application with its embedded JRE.
- **build-all.xml** will execute all the build files for all the jars, in addition to the above jars, to create the final Release ZIP file..

JSON Handler Layer - creoson-json project

Jar File: creoson-json.jar

Command Handlers

The primary entry point and API handler is **JShellJsonHandler**. It receives the requests and passes them to the appropriate handler for the command. The command handlers are **JLJson{cmd}Handler**. The comment parent class of all command handlers is **JLJsonCommandHandler**. The command handlers contain an “action” method for every function within the command family.

Help Generators

THe help-generator classes for each command are **JLJson{cmd}Help**. These classes are not used at runtime, instead they are accessed once when a developer runs the **GenTemplates** command to generate the help doc.

The help-generator classes make use of a set of template classes in **com.simplifiedlogic.nitro.jshell.json.template** to describe an API function and create examples for it.

Buildfiles

- **build.xml** will build creoson-json.jar

JSON Constants Layer - creoson-json-const project

Jar File: creoson-jsonconst.jar

Request/Response interfaces

This project exists to provide potential Java client applications with constants for the fields and values used in server calls. The same constants are used internally in the creoson-json code.

The **JL{cmd}RequestParams** contain:

- COMMAND constant to define the command value for the command group
- FUNC_* constants to define the API function names
- PARAM_* constants to define the names of request parameters
- Other constants to define special field values (such as valid parameter data types)

The **JL{cmd}ResponseParams** contain:

- OUTPUT_* constants to define the names of response parameters
- Other constants to define special field values (such as valid parameter data types)

Buildfiles

- **build.xml** will build creoson-jsonconst.jar

JSON/JShell interface layer - creoson-intf project

Jar File: creoson-intf.jar

Command Handler Interfaces

The command interfaces for each command group are **IJL{cmd}** . They define the internal JShell API methods that are exposed to the outside. Intended to be more useful if there were alternate implementations of the JShell API.

Shared Data Objects

These are low-level POJOs which are used to pass more complex data between the JSON and JShell layers, either as request or response data.

Buildfiles

- **build.xml** will build creoson-intf.jar

JShell layer - creoson-core project

Jar File: creoson-core.jar

Command Handlers

These perform the primary work of interacting with the JLink library and implementing the API functions. These classes are **JL{cmd}** and are in **com.simplifiedlogic.nitro.jlink.impl** .

Accesses to these classes from the outside is provided by the **JShellProvider** class as an entry point for this library.

Loopers

Found in **com.simplifiedlogic.nitro.util**, these are classes which loop through a list of Creo objects (such as parameters on a model), filter them based on one or more properties (like name or status) and call a subclass-defined method to perform an action on each item. Much like the Pro/TOOLKIT "Visit" functions. These classes are named **{object type}Looper**, and their subclasses are usually declared as inner classes in the Command Handlers.

Utility classes

Various classes that contain static utility methods (such as **JlinkUtils**) or class that specialize in data conversion (such as **JLMatrixMaker**). They are scattered in various packages.

"Call" JLink wrapper classes

Rather than call JLink methods directly, all classes in this library go through wrapper methods to get to JLink. With the exception of some type classes like **ModelItemType**, there are no references to JLink classes anywhere else in the JShell code.

The reason for the wrapper classes is so that we could add debugging to show when JLink methods were being called, and perhaps how long it took them to execute.

Each wrapper class wraps exactly one JLink class, and it mirrors it in name and location. Not all methods are mirrored; only those which are needed by the other JShell classes. Any new JLink calls that are added may require adding new wrapper methods to these wrapper classes.

Ideally there should be no extraneous code in the "Call" classes, just the minimum necessary to wrap the JLink. That way it would be fairly easy to get rid of the "Call" classes and convert the rest of JShell to use JLink classes directly.

Example of the wrapping:
- The JLink class **com.ptc.pfc.pfcModel.Model**
- Is wrapped by JShell class **CallModel**
- Which is found in package **com.simplifiedlogic.nitro.jlink.calls.model**
- And contains methods such as **getGenericName()** which wraps the JLink method **GetGenericName()**.

Buildfiles

- **build.xml** will build creoson-core.jar

Creo-only functionality - creofuncs project

Jar File: creofuncs.jar

Special classes

Originally JShell was written to be compatible with Pro/ENGINEER and Creo both. However, some JLink methods and classes were not introduced until Creo came along, and are incompatible with Pro/ENGINEER (Wildfire).

This library acts like a plug-in for the creoson-core library, and contains classes that handle these Creo-only calls. They are called via reflection from the creoson-core library, and if they fail to load then the user gets an error that the functionality is not available.

Buildfiles

- **build.xml** will build creofuncs.jar