Apollo

Overview specification

Confidential

Revision number: 1

Date:

Name: P. van der Velde

# Disclaimer

This specification is not, by any stretch of the imagination, complete. It will need to be revised several times before it is complete. Currently several major parts are either missing or incomplete. This disclaimer will be updated to reflect any change in these sections. Finally a specification is supposed to be a ‘living’ document and therefore never complete. What were you thinking; better learn to live with this fact.

# Introduction

This specification will define the global architecture of the Apollo application. The Apollo application will consist of three different parts, the core, the user interface and one or more plug-ins. The core parts provide the administration of the plug-ins and the user interface components. Furthermore these parts also provide the communication methods for the different sections of the application. The plug-ins will contain the data generation, storage and visualization components.

# Architecture

The architecture of the Apollo application describes the individual parts, the core, the user interface and the plug-ins, and the manner in which these parts cooperate. The first part that will be described is the architecture of the core system, followed by the architectures of the user interface and the global architecture of the plug-ins.

## Core

The main goal for the core is to provide the Apollo application with the basic utilities necessary to function. The core holds the kernel, responsible for the low level application functions, the project system and the user interface connections. The main responsibilities of the core are:

* Provide a communication pathway between different parts of the application. The parts in question are:
  + Licensing
  + Logging
  + Persistence
  + Plug-in storage
  + Project
  + User interface
* Loading the different parts in the correct order upon application start-up and unloading them upon application shutdown.
* Provide serialization capabilities for every component that needs to write information to a persistent store.
* Tracking and verifying license information.
* Tracking and maintaining configuration data.
* Providing auto-update capabilities.

Besides providing the infrastructure to achieve the before mentioned responsibilities the core also provides the connection points for external parts being the UI and the project plug-ins. Separate API’s are available for both parts thus providing them with the ability to connect to the Apollo application.

## User interface

The main goal for the user interface is to provide the user of the Apollo application with a set of controls which allow control over the actions of the Apollo application. The user interface(s) connect to the core of Apollo via the User Interface Layer which resides in the core of Apollo.

* What does the UI do?
* Which UI clients are there
  + CAD plugin
  + Project explorer
  + Server / peer-2-peer node (tentatively)
  + Command line (tentatively) (used to read scripts only)

## Plug-ins

* What is expected from the plug-ins (see the plug-in specification doc for more specific details)?
* What can plug-ins do?
* What can plug-ins not do?
* What are the rules for plug-ins?
* What area’s can be extended? Suggestion:
  + Experiment sub-elements
  + Data processing and visualization
  + Data import and export
  + Additional experiment types

## Interaction

* The core provides general services that can be used by both the UI and the plug-ins. The UI will have access to more different services than the plug-ins will.
* The core provides several different external API’s that can be used
  + User interface API. This API provides a way for the UI to interact with the core parts of Apollo.
  + Generic plug-in API. This allows components to have their own plug-ins. Defines a series of standard rules that allow the core to locate and classify plug-ins.
  + Project plug-in API. Allows developers to write project plug-ins. Project plug-ins provide extra types of generators and visualizers(?). This API is based on the generic plug-in API.
  + Component plug-in API. Allows developers to write components. The plug-in API ensures that plug-ins can be found by the system.

Start up

* Done by the bootstrapper. Will start the service provider which loads the services.
* Bootstrapper also check license information. If the license check fails then the UI will still be started but in the limited trial mode. Furthermore a message will be shown indicating that the system is running in trial mode.

Shut down

* Started by the UI (user requested) or due to an app wide error / exception
* All services are notified of shutdown. Shutdown cannot be stopped / cancelled (if there are unsaved files they need to be saved before the actual shutdown process is started).
* Shutdown will stop all the services, running the dependency tree backwards (starting with the UI). Once all the services are stopped the individual AppDomains are killed and the application exists.