Europa$_2$ Quick Start Guide

September 2005

europa-help@nx.mail.arc.nasa.gov

Nasa Ames Research Center,

Moffett Field, CA 94035

Table of FIGURES

1 Introduction

This guide will take you through the installation of Europa$_2$ and familiarize you with its operation via a small application based on a planetary rover. It is structured as follows

- Section 2: lists the software that must be installed on your system before Europa$_2$ can be installed. The time required by this stage depends upon how much of the perquisite software is already installed on your system. You may require the assistance of a systems administrator.

- Section 3: describes the Europa$_2$ installation and testing process. This stage should take around one hour and requires some simple editing of environment configuration files.

- Section 4: introduces the operation of Europa$_2$ through a small application based upon a planetary rover domain. It will take approximately two hours to work through this example.

- Section 5: provides stand alone check lists for the environment variable and software perquisites together with a list of frequently asked questions that our user community has raised while reading this guide.

- Section 6: lists the frequently asked questions raised by users during the installation of the system.

This quick start guide is supported by the following detailed documentation. You must first install and build the system before you can access this documentation.

- **Users' Guide** provides a tutorial for building Europa$_2$ applications and includes detailed documentation for each construct in our modeling language.

- **Developers' Guide** is designed for people who want to extend Europa2's functionality rather than applying the system to a application (users' guide's purpose). It describes the overall software architecture and documents each software component in the system.

2 Installation

We will step through the installation of the system in this section. Europa$_2$ consists of two major components: the PLASMA plan database engine and the PlanWorks user interface. We will build and test the PLASMA engine first as the PlanWorks interface depends upon it.

2.1 Software requirements

We require that you use one of the supported operating systems and have the following compilers, build tools, documentation generation tool, and relational database installed before using Europa$_2$. Note that the MySQL database is only used by PlanWorks You may omit it if you plan to use only PLASMA.

2.1.1 Operating System

Europa$_2$ is built and tested on the following platforms

- Red Hat Linux 2.4
- OS X 3.2 (Jaguar), 3.3 (Panther), and 3.4 (Tiger)
- Solaris 5.8

2.1.2 Compilers

Compiler software is required to translate the Europa$_2$ source code shipped to you into an executable application. Europa$_2$ is composed of C++ and Java source code.

- gcc 3.3.3 to gcc 4.0
    - gcc ships with our supported operating systems. Please see your system documentation for details on how to install it.
- Java 1.42 or higher.
    - Java can be downloaded from http://java.sun.com/

2.1.3 Build tools

Build tool software is required to orchestrate the compiler software in building the system. The PLASMA engine uses the *Jam* build tool rather than the more common *make* tool. Jam is open source and freely available.

- Jam 2.4 or higher
  - http://www.perforce.com/jam/jam.html

The PlanWorks interface uses the *Ant* build tool. Ant is a free tool provided by the apache project.

- Ant
  - http://ant.apache.org/

2.1.4 Documentation Generation Tool

Europa$_2$ uses the *doxygen* tool to build its hyperlinked documentation

- doxygen 1.2.16 or higher
  - http://www.doxygen.org/

2.1.5 MySQL Relational Database

The PlanWorks interface requires that the MySQL database server be installed. This is an open source database available for all the operating system platforms that support Europa$_2$.

- MySQL version 4.013, 4.018, 4.1.1, and 4.1.7
  - http://www.mysql.com/

2.2 Unpacking the Europa$_2$ software

Europa$_2$ ships as a single file that must be unpacked before it can be used. Using your operating systems command line shell change directory to the location where you want the system installed then enter the following two commands in sequence (substitute <version> for the actual version number shipped to you).

```
gzip –d europa2-<version>.gzip
tar –x europa2-<version>.tar
```

Note the location where you unpacked the system. We will refer to this as your *install-directory* throughout this guide. Your install-directory will contain two directories: PLASMA for the PLASMA engine and PlanWorks for the PlanWorks user interface.

2.3 Accessing the Documentation

[The other documentation is still in progress]

Enter the Documentation directory and type “jam documentation”. Go into the html directory produced and select ???

2.4 Configuring your Software Environment

This section details the software environment variable settings required by Europa$_2$. It assumes that you are familiar with Unix shells. Please see your operating system's documentation for details. All examples use cshell but are readily adapted to other shells.

2.4.1 Setting your PATH

Your *PATH* variable must include the locations of your Ant, gcc, Jam, Java, and Doxygen installations. If you don't know the locations, you can use the find commands listed below to help locate the executables. Then look for the directory the actual executable is in. Often, but not always, this is a "bin" directory.

```
find /usr -name ant*
find /usr -name gcc-*
find /usr -name 'j2sdk1.*' -o -name 'jdk1.*'
find /usr -name 'jam2.*'
find /usr -name doxygen
```

2.4.2 Setting your LD_LIBRARY_PATH

Your *LD_LIBRARY_PATH* (DYLD_LIBRARY on the Mac) must include the locations of your gcc and the Europa$_2$ shared libraries. The shared libraries are relative to the location where you unpacked the system (section 2.2).

```
<install-directory>/PLASMA/lib
```

2.4.3 Setting your JAVA_HOME

Your *JAVA_HOME* must point to the location where you installed Java

2.4.4 Setting your OSTYPE

You must define a OSTYPE environment variable for the PlanWorks interface. Darwin is the name used for the Apple OS X operating system.

```
OSTYPE = linux | solaris | darwin
```

2.4.5 Setting your PLANWORKS_HOME

You need a PLANWORKS_HOME variable that points to the directory where you installed PlanWorks.

```
PLANWORKS_HOME <install-directory>/PlanWorks
```

2.4.6 Example .cshrc file settings

The following fragment is taken from a Europa$_2$ user's .csrhc file. It illustrates the necessary settings but is specific to the actual locations where the software was installed on this person's computer. It is included only as a guide to assist you.

```
setenv JAVA_HOME /usr/java/j2sdk1.4.2_04
setenv JAM_HOME /usr/local/jam2.5
setenv GCC_HOME /user/local/gcc-3.3.3/bin
setenv ANT_HOME /usr/local/apache-ant/

set path = ( ${path} $JAVA_HOME )
set path = ( ${path} $GCC_HOME
set path = ( ${path} $JAM_HOME )
set path = ( ${path} $ANT_HOME )

setenv LD_LIBRARY_PATH /usr/local/gcc-3.3.3/lib
setenv LD_LIBRARY_PATH
"$LD_LIBRARY_PATH":/home/uname/europa/PLASMA/lib

setenv OSTYPE Darwin
setenv PLANWORKS_HOME /user/uname/europa2/PlanWorks
```

2.4.7 Verify settings with "checkreqs"

We provide a script for automatically checking that the environment settings have been made correctly. Enter the following command into your operating system's command line shell.

```
<install-directory>/PLASMA/checkreqs
```

The script will report if there are problems with the above settings. Please return to the relevant section and check for typographical errors and confirm that the paths you have entered do lead to the desired file

2.5 Building the PLASMA Engine

We are now ready to build the system. Enter the following command into your operating system's command line shell.

```
<install-directory>/PLASMA/jam
```

This will initiate the building process that will take approximately one hour depending on the speed of your computer. The following is a small fragment of the trace information you should see as the system builds.

```
/users/uname/europa2/PLASMA > jam
...patience...
...found 1168 target(s)...
...updating 563 target(s)...
MkDir1 lib
MkDir1 Utils/objects
C++ Utils/objects/LockManager_g_pic.o
C++ Utils/objects/Debug_g_pic.o
C++ Utils/objects/LabelStr_g_pic.o
C++ Utils/objects/Entity_g_pic.o
C++ Utils/objects/Error_g_pic.o
…
…
…
Chmod1 System/test/gnats_2572_interval-tx_g_rt
C++ System/test/stackGenerator.o
Link System/test/stackGenerator
Chmod1 System/test/stackGenerator
...updated 563 target(s).
```

Figure 1: Trace of PLASMA Engine build ouptut

Problems are reported as "failed targets." If this occurs please rerun the *checkreqs* script and verify your installation settings. If the system still will not build the please contact the Europa$_2$ team using the e-mail address on the front of this guide for assistance

2.6 Testing the PLASMA Engine

We are now ready to test that the newly built system is operating correctly. Enter the following command into your operating system's command line shell.

```
<install-directory>/PLASMA/jam run-install-tests
```

The tests should take around ten minuets to run depending on your computers performance. There should be no errors reported. Contact the Europa$_2$ team using the e-mail address on the front of this guide for assistance

2.7 Building the PlanWorks interface

You must copy the following files from your MySQL installation into your PlanWorks installation.

- Copy the binary files from your mysql install to *<install-directory>/PlanWorks/lib/mysql/bin*.
- Overwrite the files in *<install-directory>/PlanWorks/lib/mysql/share/mysql/English/* with the files in *path-to-your-mysql/share/mysql/English.*

Enter your PlanWorks directory from your operating systems command line shell and type the following command to initialize the systems database. This only has to be once for an installation.

```
ant planWorksDbInstall
```

Next enter the following command to create the JNI (Java Native Interface) for controlling the PLASMA engine. This only has to be done once for an installation.

```
ant createJNI
```

Now build and run the PlanWorks tool with the following command.

```
ant
```

2.8 Testing the PlanWorks interface

Typing *ant* within the PlanWorks directory will display the application screen shown in Figure 2 if PlanWorks is correctly installed. For the moment simply exit PlanWorks using the *File* menu at the top left menu bar.

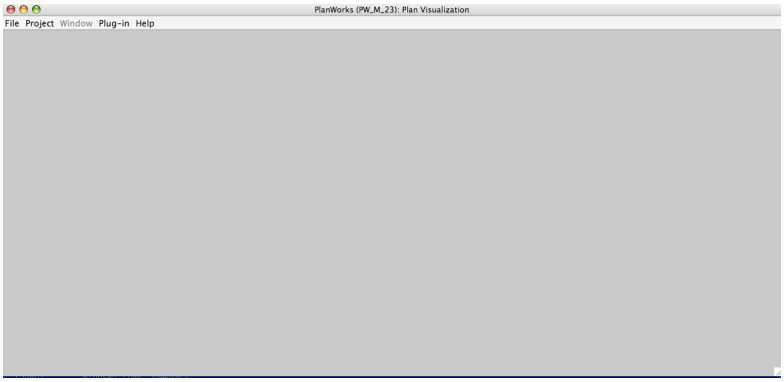

Figure 2: PlanWorks initial screen

Congratulations! You have installed and tested Europa$_2$!

3 Operation

There are two primary interfaces for interacting with Europa$_2$. The *command-line interface* works directly from the operating system command line shell and will allow you to rapidly run tests and build the system. The *PlanWorks* interface is a graphical user interface for running the system and inspecting the plans it generates.

We will use a simple planetary rover application to introduce the interfaces to Europa$_2$. We begin by briefly giving an overview of this domain before describing each interface in turn beginning with the command-line.

3.1 Overview of the Simple Planetary Rover Application

The Mars Exploration Rover (MER) mission that is operating the *Spirit* and *Opportunity* rovers on the Martian surface motivates this domain[1]. Our simple rover is inspired by this mission and has the following components:

- A navigator module that is responsible for controlling the position of the rover on the planet's surface. The rover may move between two points providing the terrain permits it and that it has the power available to make the movement. The greater the distance between points and the change in elevation the greater the power requirements.
- A simple instrument on a robotic arm that can be placed against a rock to measure its properties.
- A communication system that allows the rover to transmit the data obtained from the simple instrument to either the rover's lander or directly to earth.
- The lander that delivered the rover to the planet's surface also contains a more powerful communication relay than is available on the rover itself.

Figure 3 presents a sketch of these components.

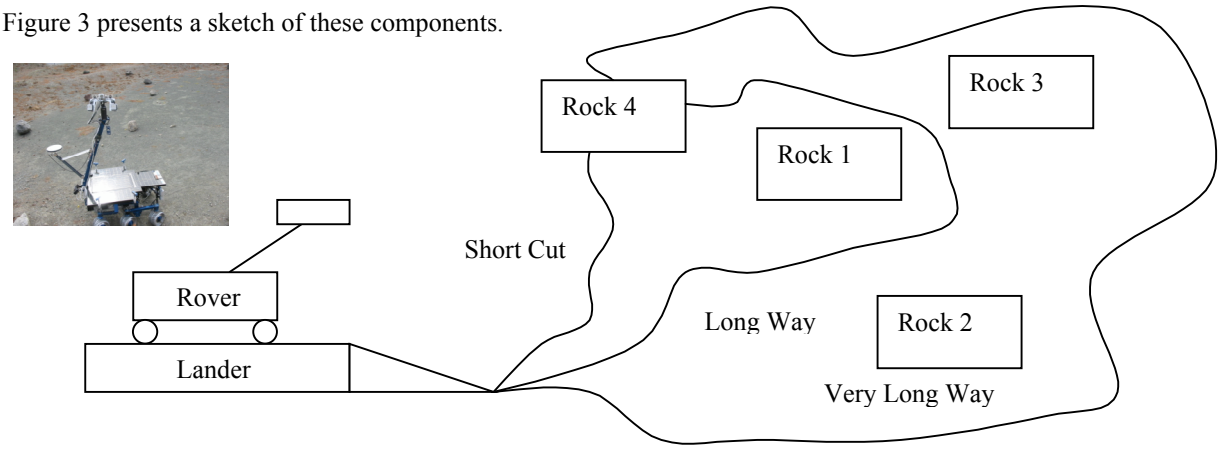

Figure 3: Sketch of Planetary Rover Application

[1] MER homepage: http://marsrovers.jpl.nasa.gov/home/

There are two components to the Europa$_2$ application. The *application domain model* or simply the *domain model* describes the components of the rover and its environment. The *initial state* file specifies a planning problem instance in terms of the state in which the world is initially and the changes to that state we want our plan to bring about.

We describe these two components in our modeling language, *NDDL* – New Domain Modeling Language. Open the following NDDL files in a text editor (e.g. emacs):

- Model: <install-directory>/PLASMA/Examples/SimpleRover/SimpleRover-model.nddl
- Initial state: <install-directory>/PLASMA/Examples/SimpleRover/SimpleRover-initial-state.nddl

We will examine the model file then the initial state.

3.1.1 SimpleRover-model.nddl

Figure 4 presents the definition of our planetary rover from the domain model file. The rover is made-up of three concurrent threads of execution or *timelines* and a single resource (battery). The *commands* timeline handles the taking of samples and the communication of the resultant data back to earth. The *navigator* timeline manages the position of the rover on the planet. The *instrument* timeline handles the positioning of the rover's sensor on rocks so that samples can bet taken. The *battery* resource manages the power required for activity on the other timelines

```
class Rover {
 Commands commands; // High-level rover commands
 Navigator navigator; // Handles position control and achievement
 Instrument instrument; // Handles rock sampling
 Battery mainBattery; // Power for other timelines

 Rover(Battery r){
  commands = new Commands();
  navigator = new Navigator();
  instrument = new Instrument();
  mainBattery = r;
 }
```

Figure 4: Class Rover

Figure 5 shows the definition of the navigator timeline class of the rover. This class contains the two states (or predicates) *At* and *Going*. A timeline may only execute one predicate at a particular time. This means that our rover is either at a location or is navigating between a pair of locations. The detailed constraints in the At predicate specify that an At predicate must be met by a going predicate and must meet a going predicate. The *eq* location constraints specify that the incoming navigate predicate must have its to location set to this At point while the outgoing going must have its from predicate set to this location.

Figure 6 defines the battery resource of the rover. It states that a rover battery has an initial capacity (*ic)* and has a minimum power level (*ll_min*) and a maximum power level (*ll_max*).

You are encouraged to spend a little time examining the other elements of the model file. It takes a little time to understand NDDL domains and reading through a file several times is a great way to proceed.

```
class Navigator
{
 // Rover maybe AT a location.
 predicate At{
  Location location;
 }

 // Rover maybe going between two locations.
 predicate Going{
  Location from;
  Location to;
  neq(from, to);
 }
}

Navigator::At{
 met_by(object.Going from);
 eq(from.to, location);
 meets(object.Going to);
 eq(to.from, location);
}

Navigator::Going{
 met_by(object.At _from);
 eq(_from.location, from);
 meets(object.At _to);
 eq(_to.location, to);

 // Select a path from those available between the 2 points
 Path p : {
  eq(p.from, from);
  eq(p.to, to);
 };

 // Pull power from the battery equal to the path cost.
 starts(Battery.change tx);
 eq(tx.quantity, p.cost);
}
```

Figure 5: Navigator Class

```
class Battery extends Resource {
  Battery(float ic, float ll_min, float ll_max){
   super(ic, ll_min, ll_max, 0.0, 0.0, MINUS_INFINITY,
             MINUS_INFINITY);
  }
}
```

Figure 6: Battery Class

3.1.2 SimpleRover-initial-state.nddl

Figure 7 presents the contents of the SimpleRover-inital-state file that defines a specific planet situation and sampling goal that we need to plan for. The *PlannerConfig* line is just bounding the amount of time that we will allow our planner to look for a solution. In this case it make take up to 100 planning steps and not more than 600 seconds. If a solution cannot be found within those bounds the planner will return failure.

```
PlannerConfig world = new PlannerConfig(0, 100, 600);

Location lander = new Location("LANDER");
Location rock1 = new Location("ROCK1");
Location rock2 = new Location("ROCK2”);
Location rock3 = new Location("ROCK3");
Location rock4 = new Location("ROCK4");

// Allocate paths
Path p1 = new Path("Very Long Way", lander, rock4, -2000.0);
Path p2 = new Path("Moderately Long Way", lander, rock4, -
1500.0);
Path p3 = new Path("Short Cut", lander, rock4, -400.0);

// Allocate Rover
Battery battery = new Battery(1000.0, 0.0, 1000.0);
Rover spirit = new Rover(battery);

// Establish the initial position for spirit
goal(Navigator.At initialPosition);
eq(initialPosition.start, world.m_horizonStart); // What time -
the start of this planmning horizon

eq(initialPosition.location, lander);

// Establish the goal - to take a sample at a position
goal(Commands.TakeSample sample);
sample.start.specify(50);
sample.rock.specify(rock4); // Want to get to rock4

// Establish the initial instrument state
eq(stowed.start, world.m_horizonStart);
```

Figure 7: SimpleRover-inital-state.nddl

The planet has five locations – four rocks and a lander. There are three paths connecting the *lander* directly to *rock4*. The paths vary in the amount of battery power required to transverse them. We declare a resource called *battery* with an initial and maximum capacity of 1000 units. We define a single rover, *spirit*, with this battery.

We define the initial position of the rover as the lander. We define the rover’s goal as to take a sample at *rock4*. Finally we record that the sampling instrument is initially stowed.

3.2 Interacting from the command-line interface

Enter the following command into your operating sytem's command line shell from the SimpleRover diretory to invoke the PLASMA engine on the SimpleRover domain:

- Jam SimpleRover

The plan generated by the system will be output in the file

- RUN_SimpleRover-planner_g_rt.SimpleRover-initial-state.xml.PlannerConfig.xml.output.

Figure 8 shows the element of this output file that describes the actions on the navigator timeline of our rover during the plan. Activity on a timeline is divided into *Token*s that correspond to instances of the predicates defined in the model defined in Figure 5. Each token is bounded by a temporal interval. The *Navigator.At* token starts and time 0 and finishes at time 1. The *Navigaor.going* token begins at time 1 and can finish anytime between time 2 and time 32.

```
Navigator:spirit.navigator************************
      Tokens ************************
            [ INT_INTERVAL:CLOSED[0, 0] ]
Navigator.At(location=Location:CLOSED{lander})
            [ INT_INTERVAL:CLOSED[1, 1] ]
            [ INT_INTERVAL:CLOSED[1, 1] ]
Navigator.Going(from=Location:CLOSED{lander}
                to=Location:CLOSED{rock4})
            [ INT_INTERVAL:CLOSED[2, 32] ]
            [ INT_INTERVAL:CLOSED[2, 32] ]
Navigator.At(location=Location:CLOSED{rock4})
            [ INT_INTERVAL:CLOSED[62, +inf] ]
      End Tokens ********************
End Navigator:spirit.navigator************************
```

Figure 8: Navigator Timeline Plan

You are encouraged to explore the other time lines and follow the plan to achieve the goal of sampling rock4 as defined in Figure 7.

3.2.1 Tracing

The PLASMA engine can generate extensive tracing information as it reasons. Tracing for the SimpleRover application is controlled using the *Debug.cfg* file in the application's directory. Open this file with a text editor and remove the comment ("#" character) from the :*Solver* line. Run the PLASMA engine with t he following command line and note the tracing information displayed.

- `SimpleRover-planner_g_rt SimpleRover-initial-state.xml PlannerConfig.xml`

3.3 Interacting from the PlanWorks interface

We must first instruct PlanWorks to use the configuration file in the SimpleRover application directory. Create the following environment variable:

- `setenv PPW_CONFIG install-directory/PLASMA/Examples/SimpleRover/PlanWorks.cfg`

The default settings in the PlanWorks.cfg file instruct PlanWorks to output every step of the planning process. The logger seriously slows the planner's performance, as it must write large amounts of information to disk at each step in its reasoning process. To reduce the amount of tracing information generated you may adjust the following variables within the PlanWorks.cfg file.

- `set 'WriteFinalStep=1'`
- `set 'AutoWrite=0'`

We must also tell PlanWorks were we want plans written to disk. Set the PlanWorks.cfg as follows:

- `Set WriteDest=<install-directory>/PLASMA/Examples/SimpleRover/plans`

3.3.1 Viewing a Plan Sequence

With PlanWorks configured we are now ready to start the system and create a project for a SimpleRover application. Start PlanWorks by following these two steps.

- `cd to <install-directory>/PlanWorks`
- `ant`

Once in PlanWorks select "Project--->Create" from the top level menu bar. This will bring up a Create Project window. Type "SimpleRover" in the "name" field (any name will work). In the "working directory" field, enter the full path of the directory that contains your plan sequences ("*<install-directory>/PLASMA/Examples/SimpleRover/plans*"). Click on the "Enter" button. A "Select Planning Sequence Directory" window will appear. Sequence directory(ies) created for your plan by the PartialPlanWriter are found in this directory. Select a plan directory and click the "OK" button.

Once a PlanWorks project has been created and a sequence directory has been selected you are ready to visualize your plan. Select PlanningSequence" from the top level menu bar. Now select your sequence, if you have been using the names in this example, the name of your sequence will be "SimpleRover-model". Click on this.

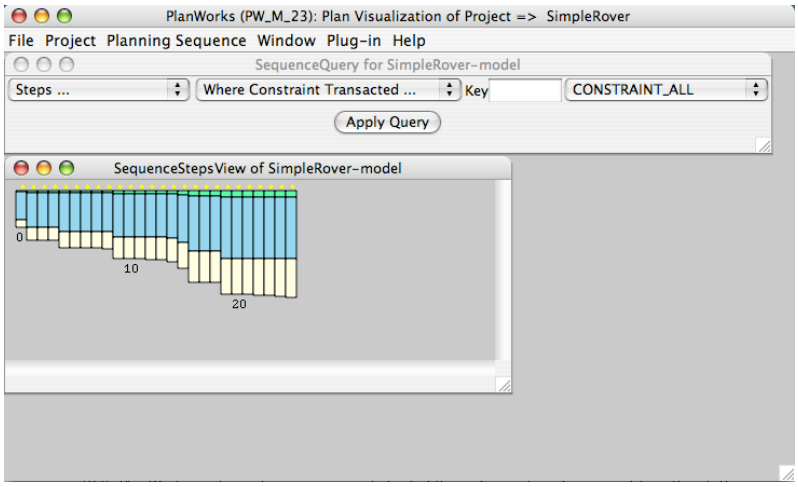

Figure 9: SimpleRover Sequence

Figure 9 shows the window called "SequenceStepsView of SimpleRover-model" that should appear. Note the sequence of vertical bars in the window. Each vertical bar represents a partial plan. If the planner found a plan, the vertical bar on the far right will represent your completed plan. A yellow dot above this bar indicates that planner data was logged for this step. A red dot indicates that no plan data was logged and implies that your PlanWorks.cfg file may not be properly configured. Scroll to the vertical bar on the far right and right click anywhere on this bar. A menu will appear with a list of items that you can now visualize. You can select these items one at a time or you can select them all at once with the "Open all Views" menu item at the bottom of the list. Once a view has been opened the yellow dot will turn green indicating that the data for this step has been loaded in the PlanWorks database.

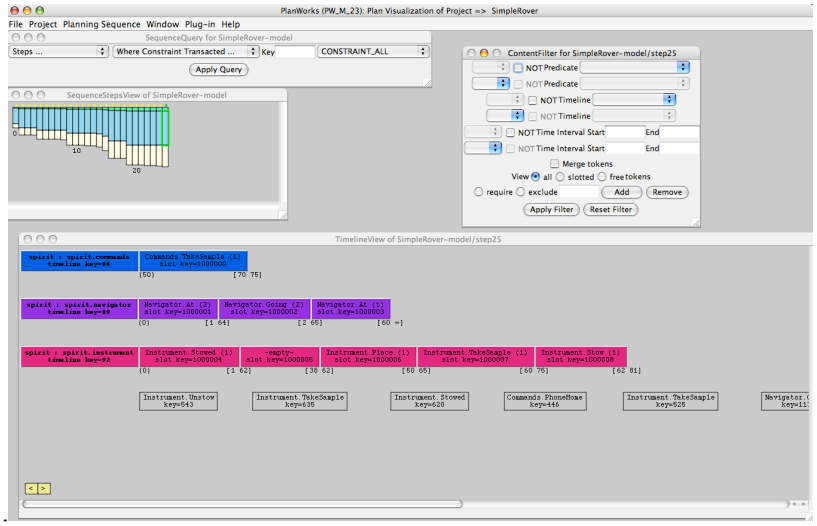

Figure 10: SimpleRover Sequence timeline view

Figure 10 shows the timeline view that you can generate by right clicking on the rightmost blue bar in the sequence view and selecting the “Open Timeline View” option.

3.3.2 Generating a New Plan Sequence

The PlanWorks interface can control the PLASMA engine and instruct it to generate new plan sequences. We must fist adjust the PlanWorks.cfg file to allows this. Exit from PlanWorks using the exit command from the file menu and opent he PlanWorks.cfg file in a text editor.

Uncomment the two lines of the RuleConfigSection and set the SourcePath to the project directory. This tells the PPW where to look for the nddl files that contain the rules source code.

Restart PlanWorks. Once in PlanWorks select "Project--->Open" from the top-level menu bar. This will bring up an "Open Project" window. Select "SimpleRover" if not already selected. Click on the "OK" button. Once a project has been opened, select "Project--->Configure" from the top level menu bar. This will bring up a "Configure Project" window with "Simple" in the "name" field. The "working directory" field should have the full path of the directory that contains your plan sequences. If not, enter it now. The browse button may be used to assist in finding the path.

In the "planner library path" field, enter the full path to the planner-shared library in PLASMA/lib. This is the Europa2 System shared library.

It is NOT the model library that was created in your SimpleRover directory.

The full path will look like:

- `<install-directory>/PLASMA/lib/libSystem_g.so` *`(.dylib on Mac)`*

In the "model library path" field, enter the full path to the model shared library in SimpleRover. This is the model shared library that was created in your SimpleRover directory. The full path will look like:

- `<install-directory>/PLASMA/Examples/SimpleRover/lib/libHelloWorld_g.so` *`(.dylib on Mac)`*

In the "model initial state path" field, enter the full path to the model initial state xml file in SimpleRover The full path will look like:

- `<install-directory>/PLASMA/Examples/SimpleRover/SimpleRover-intial-state.xml`

In the "model output destination directory" field, enter the full path to the plans directory. The full path will look like:

- `<install-directory>/PLASMA/Examples/SimpleRover/plans`

In the "planner config path" field, enter the full path and name to planner config file. It will look like:

- `<install-directory>/PLASMA/Examples/SimpleRover/PlannerConfig.xml`

Figure 11 shows the configuration setting for a particular installation on the Mac platform. Your path will be different initially but should match the PLASMA parts forward. Once all of the fields have been entered, click on the "Enter" button.

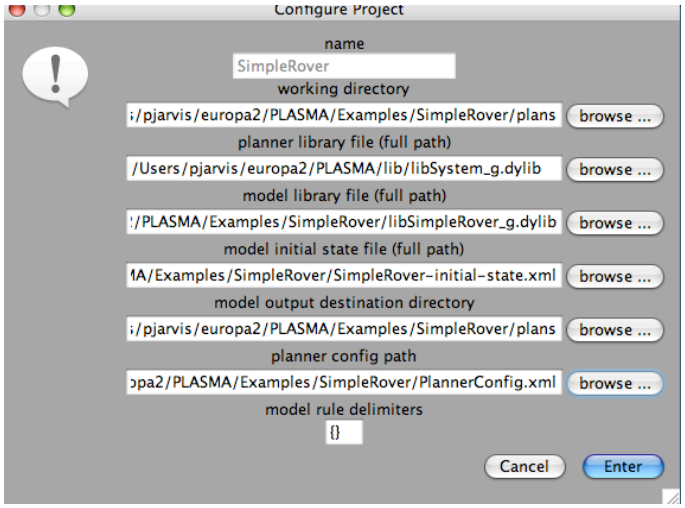

Figure 11: Project Configuration

Once a project has been configured, you are ready to run your plan fromPlanWorks. Select "Project--->New Sequence" from the top-level menu bar. This will bring up a "Create New Sequence" window. PlanWorks using the information you provided above will already fill all of the fields in. Check all fields are correct and then click on the "Start" button.

If PlanWorks successfully finds all of your required files, a "Transaction Types to Log" window will appear. (If it does not appear after a few seconds, you can check the terminal window for troubleshooting information.) The checklist of Transaction Types allows you to disable logging certain transactions in order to speed plan execution. Please lease the settings at their default values and click on the "OK" button.

A new window titled "PlannerController for HelloWorld" will come up. From this window you have the following options:

- Skip to step number N - the planner will run to step N and log data for only that step.
- Write next N steps - the planner will run for the next N steps logging data for each step.
- WriteFinalStep the planner will run to completion logging data for only the last step of the plan.
- TerminatePlanner the planner will terminate and unload the model

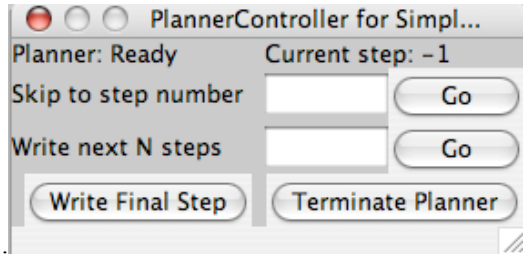

Figure 12: Planner Control Window

To try out these options enter "5" in the "Skip to step number" field and click on the adjacent "Go" button. Note that the SequenceStepsView contains six vertical bars with the last bar indicating that data was logged

for step five. You can right click on this bar and a menu will appear with items that you can now visualize for this step.

Now enter "4" in the "Write next N steps" field and click on the adjacent "Go" button. Note that four more vertical bars have appeared all indicating that data was logged. You can now right click on any of these bars to visualize the steps.

To complete the plan click on the "Write Final Step" button. The Planner status in the upper left hand corner of the PlannerController will change from "Ready" to "Running" while the PLASMA engine has control.

Once the plan completes a "Planner Status" window should come up. Unless you have modified the model it should report that the planner "Found a plan." Click on the "OK" button. The vertical bar on the far write of the SequenceStepsView represents the final plan and should have a yellow dot.Right click on this bar to visualize the final plan.

Once a plan has been run to completion or terminated, it can be rerun without exiting PlanWorks. You can make changes to your plan in a terminal window outside of PlanWorks and create new model library and initial state files using "jam libHelloWorld_g.so" in the SimpleRover directory.

To rerun a plan from PlanWorks select "Project--->New Sequence" from the top-level menu bar. All of the fields should be filled in and should be OK as long as none of the names have changed for your model. Click on the "Start" button as before. Click the "OK" button of the "Transaction Types to Log" window and a new PlannerController window will appear.

4 Checklists

4.1 Environment Variables

This is a sample set of Euorpa$_2$ environment variable settings. The actual paths will depend upon where the software is installed on your system. We have included it as a guide for you to check your settings against.

```
setenv JAVA_HOME /usr/java/j2sdk1.4.2_04
setenv JAM_HOME /usr/local/jam2.5
setenv GCC_HOME /user/local/gcc-3.3.3/bin
setenv ANT_HOME /usr/local/apache-ant/

set path = ( ${path} $JAVA_HOME )
set path = ( ${path} $GCC_HOME
set path = ( ${path} $JAM_HOME )
set path = ( ${path} $ANT_HOME )

setenv LD_LIBRARY_PATH /usr/local/gcc-3.3.3/lib
setenv LD_LIBRARY_PATH
"$LD_LIBRARY_PATH":/home/uname/europa/PLASMA/lib

setenv OSTYPE Darwin

setenv PLANWORKS_HOME /user/uname/europa2/PlanWorks
```

4.2 Software Requirements

Operating System	• Red Hat Linux 2.4 or • OS X 3.2 (Jaguar), 3.3 (Panther), and 3.4 (Tiger) or • Solaris 5.8
Compilers	• gcc 3.3.3 to gcc 4.0 • Java 1.42 or higher.
Build Tools	• Jam 2.4 or higher • Ant
Documentation Generation Tools	• doxygen 1.2.16 or higher
MySQL Relational Database	• MySQL version 4.013, 4.018, 4.1.1, and 4.1.7

5 Frequently Asked Questions

This section will be populated as we gain experience with users and encounter problems.