LSTS Software Toolchain

Oceans 2015

José Pinto <zepinto@lsts.pt>
Paulo Dias <pdias@lsts.pt>

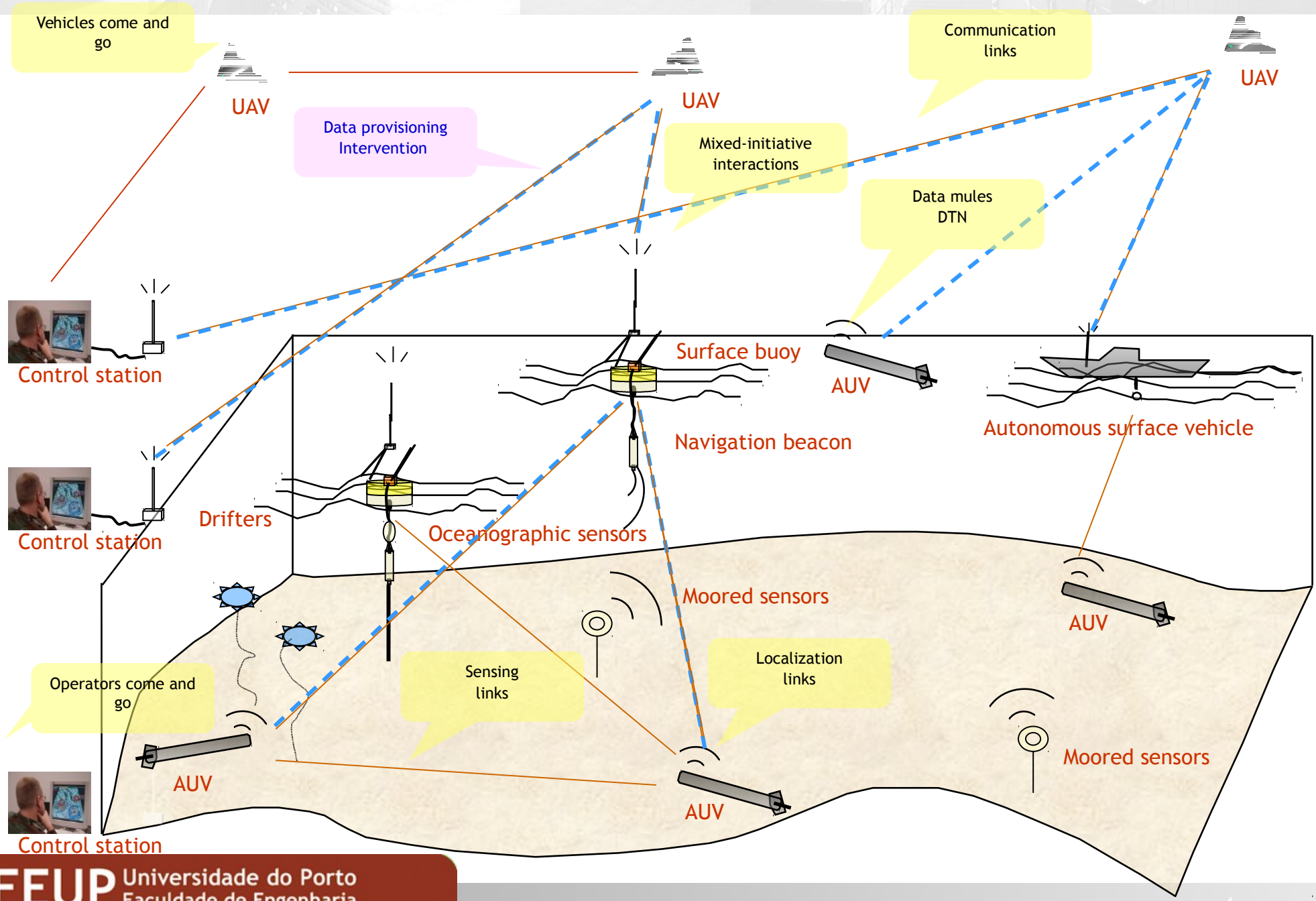
Underwater Systems and Technology Laboratory (LSTS)

Tutorial Overview

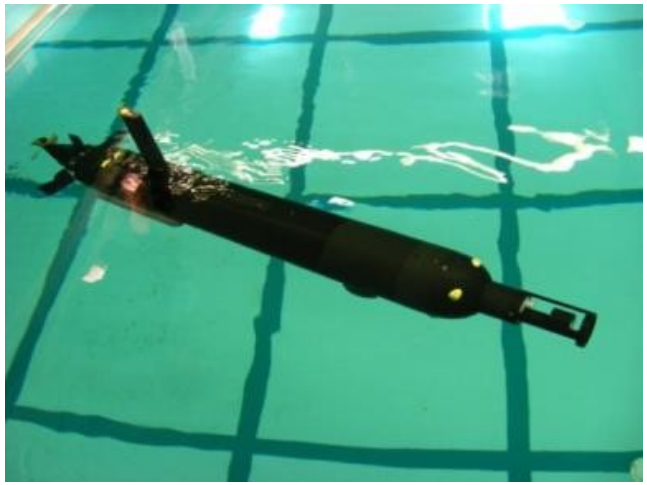
- **Background**
- **Toolchain Overview**
  - GLUED / IMC / DUNE / Neptus
- **Using the toolchain**
  - Compiling
  - Running
- **Extending**
  - DUNE, Neptus

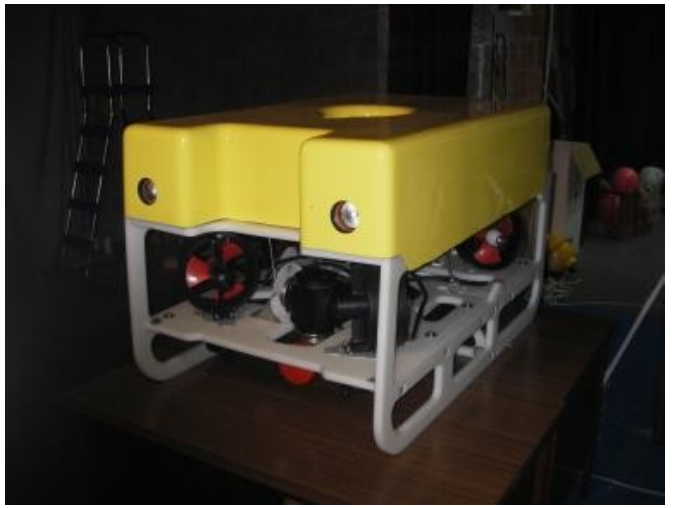
Background

Networked Vehicle Systems

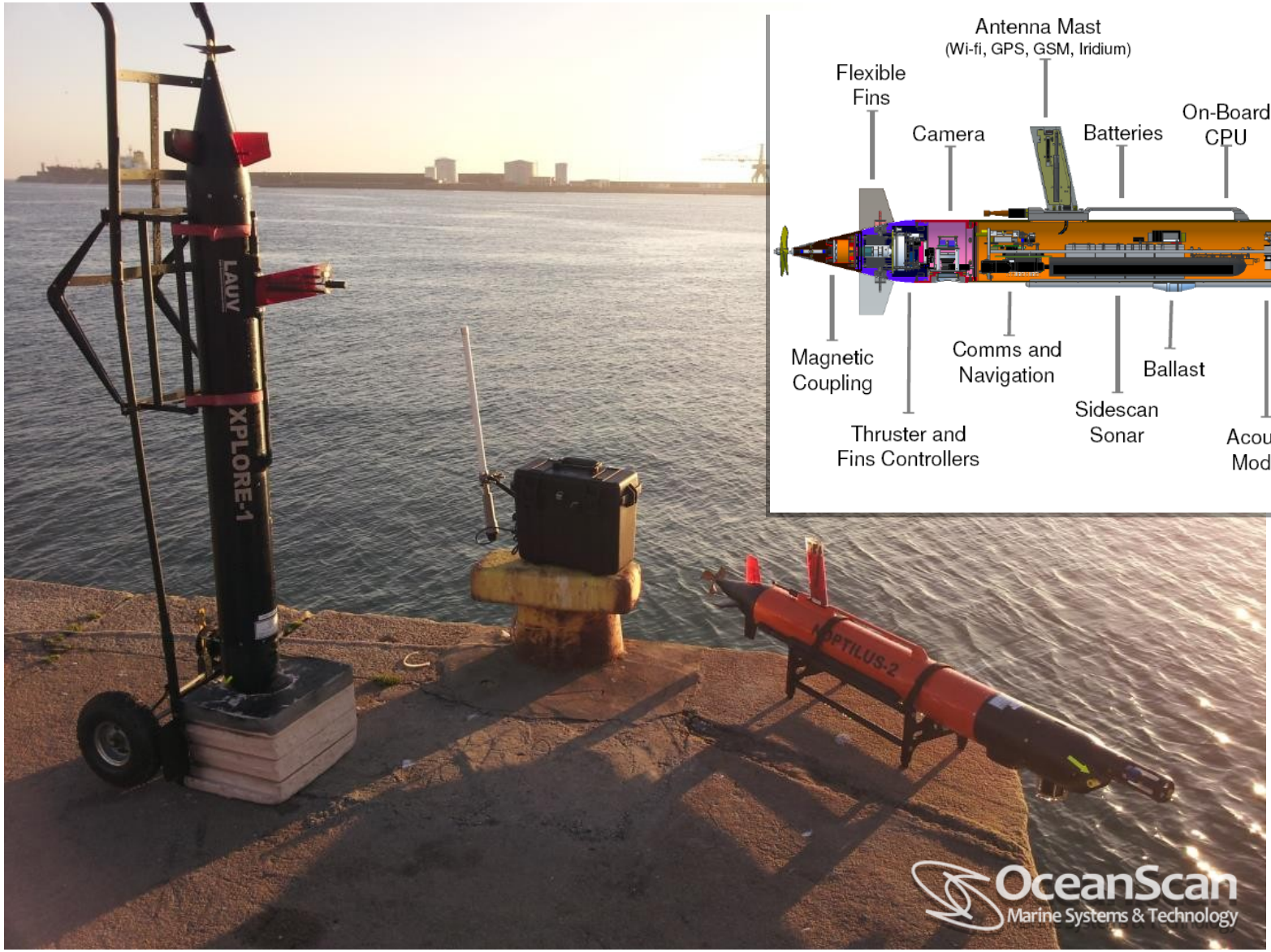


Ocean Vehicles @ Porto University





The Light AUV (LAUV)



The Light AUV (LAUV)

Endurance of up to 24 hours

Moving at ~1 m/s

1 man operation

Portable

Easy deployment and recovery

Intuitive software tools

Customizable sensor payloads

Sidescan / Multibeam Sonars

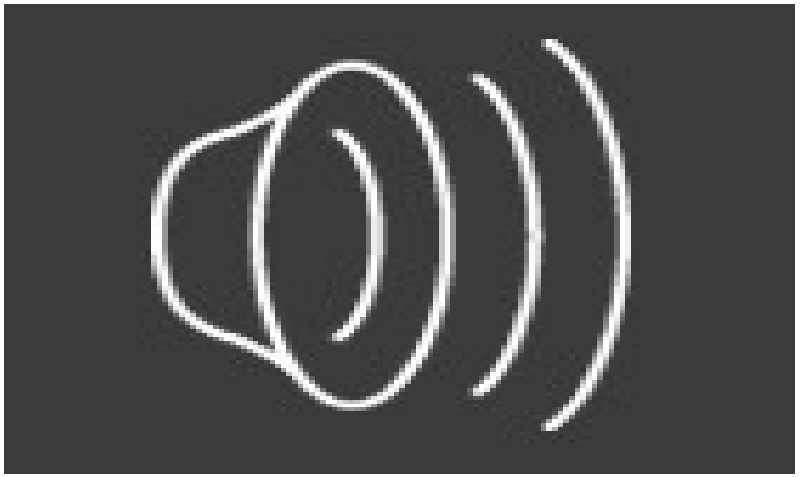
Video camera

CTD, pH, Fluorometer, ...

Other payloads

Comms: WiFi, GSM, Acoustic Modem, Iridium

Navigation: GPS, Compass, IMU, DVL, LBL, ...



CPUs

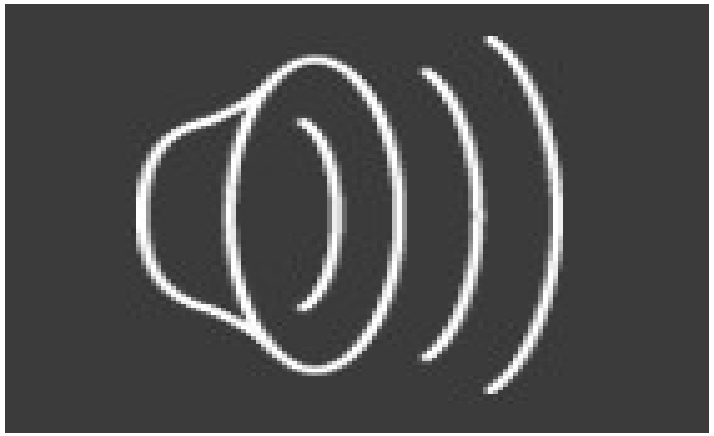
Geode X86 (Main CPU)

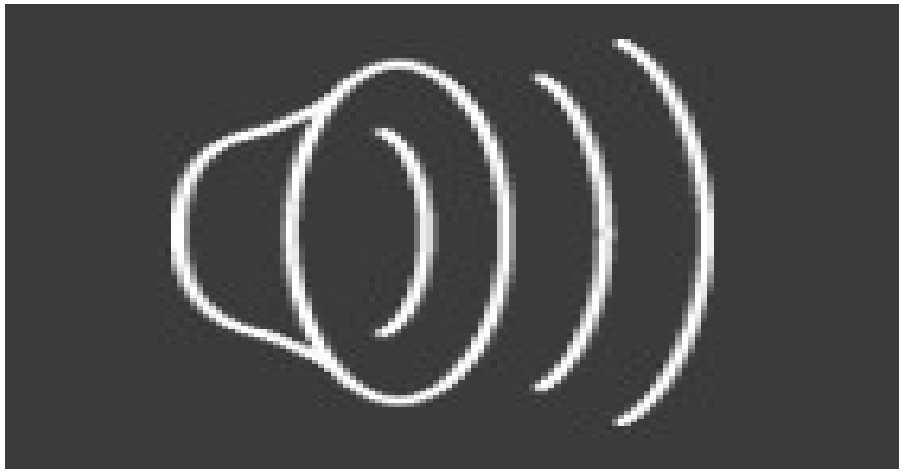
Beagle Bone Black (Auxiliary)

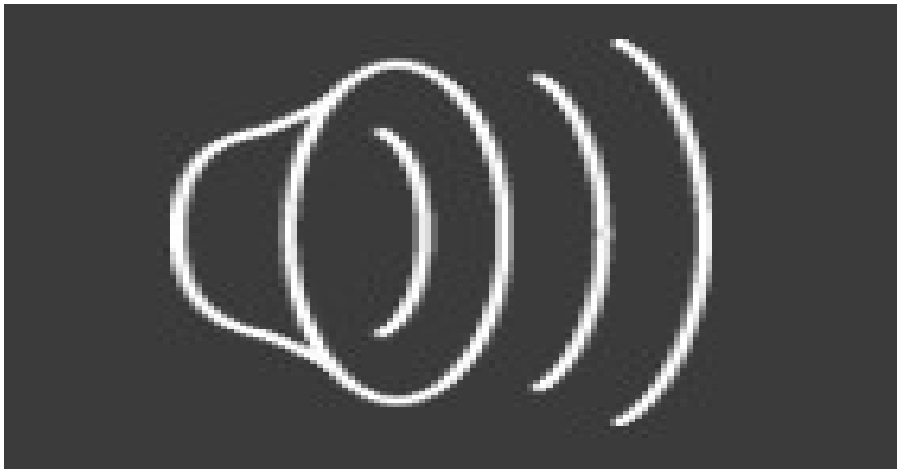
Aerial Vehicles @ Porto University

Platforms - X8 UAV

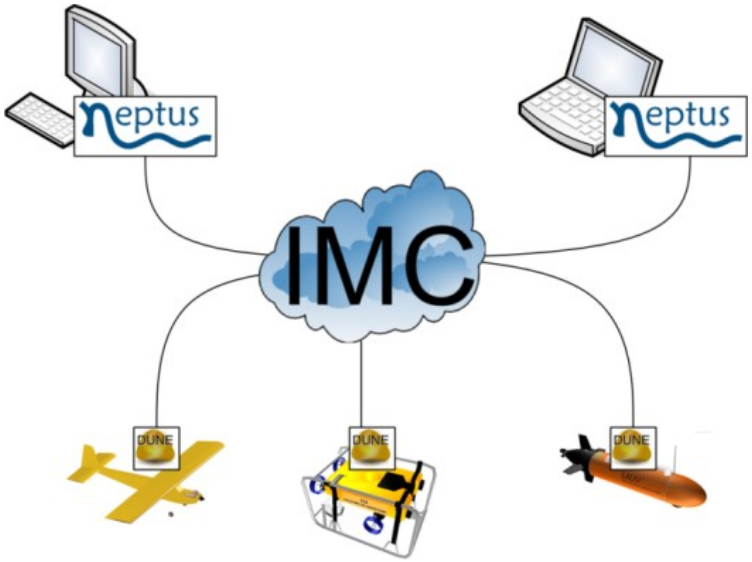
- Ardupilot + DUNE
- Planner runs on the ground
- Telemetry + video feed up to 10 Kms
  - Using auto-oriented directional antenna
- Easy deployment and recovery





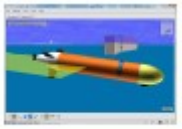


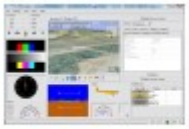
LSTS Toolchain for NVS

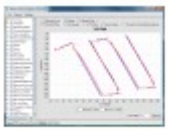


C4I – Command and Control Framework

Neptus







http://vhale.fe.up.pt

Inter Module Communications

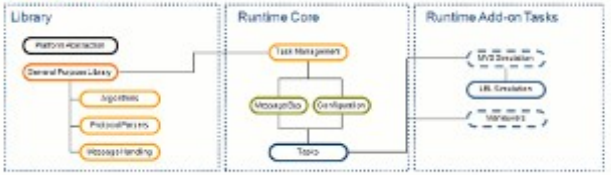
IMC

Message Protocol

http://vhale.fe.up.pt

DUNE: Uniform Navigational Environment On-board software

DUNE



http://vhale.fe.up.pt

https://www.github.com/LSTS

LSTS Timeline

- 1997 – LSTS was created
- 1997 – First AUV was purchased (WHOI REMUS AUV)
- 2004 – Neptus was created
- 2005 – First ROV built from scratch
- 2006 – DUNE and IMC were created
- 2006 – First AUV built from scratch (LAUV)
- 2007 – Projects and MoU with the PO Ministry of Defence
- ...
- 2015 – 20 LAUVs, 1 ROV, 1 ASV, 20 UAVs, 10 LAUVs sold, toolchain used by external entities (NTNU, KTH, LabUST, EvoLogics GmbH, ...)

What is It ?

- Complete software solution for autonomous vehicles
- Operating system (Linux distribution)
- API for interaction between software modules
- On-board software for sensor interaction, Control, Guidance, Navigation
- GUI Command & Control Unit
- Mission Review and Analysis (Log Analysis)

Why start from scratch ?

- The REMUS on-board software was starting to show its age (based on QNX4 and Pre ISO C++ 98)
- The code-base was developed and maintained poorly
- Freely available software had a few shortcomings:
  - Highly experimental
  - Cumbersome or impossible to use in embedded systems
  - Command & Control software was primitive
  - We disagreed with the architecture/design choices

Further motivation

To avoid things like this:

// Created by John Doe (1996)

// Updated by John Smith (1999)

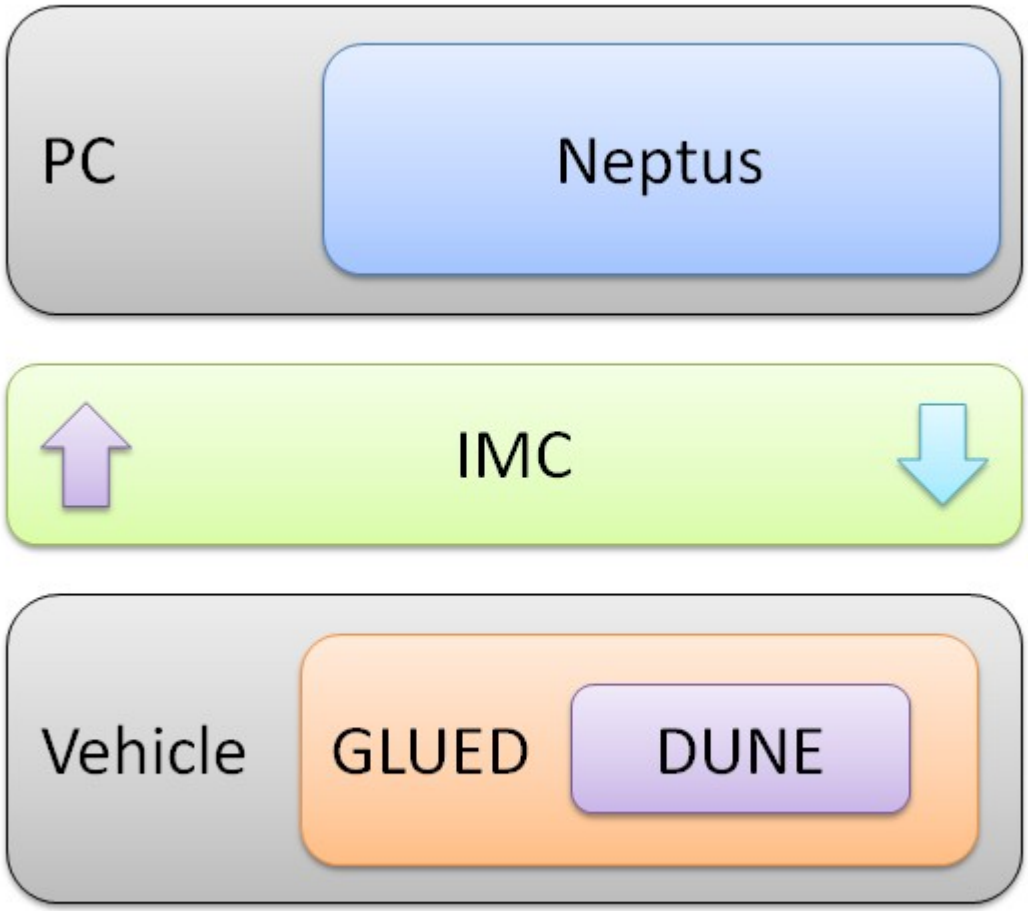
// Updated by Tommy Toe (2001)

// Vandalized by Joe Bloggs (2005)

From *common/abstraction/shared/whoi_mboard.h*

(Fictitious names)

Toolchain Overview



GLUED

GNU/Linux Uniform Environment Distribution

https://github.com/LSTS/glued

Overview

- Minimal GNU/Linux distribution focused on embedded systems
- Small footprint
  - around 10 MiB
- Fast boot time
  - 2 to 5 seconds depending on target machine and peripherals
- Target machine binaries are cross compiled (i.e., built for a platform other than the one on which the compiler is running)
- Creates a reproducible root filesystem for a given target
- Supports several x86, ARM, and MIPS targets

Motivation

- Time to build large software projects in embedded systems is almost unbearable
  - several hours vs a few minutes in modern PCs
- Embedded systems usually require bootloader and kernel customization
- The longer the system takes to boot the longer it is uncontrollable
- Upgrading the operating system should be an unattended process with a predictable outcome
- The root filesystem and target binaries should be easily replicated and traceable

Supported Hardware

- ARM
  - BeagleBone White & Black (TI AM3359 @ 1 GHz)
  - ISEE IGEPv2 (TI DM3730 @ 1 GHz)
- x86
  - IEI PM-LX 800 (AMD Geode LX @ 500 MHz)
  - IEI PM-LX2 800 (AMD Geode LX @ 500 MHz)
  - Kontron pITX (Intel Atom Z510 @ 1.6 GHz)
- MIPS
  - Ubiquiti RouterStation (Atheros AR7161 MIPS 24K @ 680 MHz)

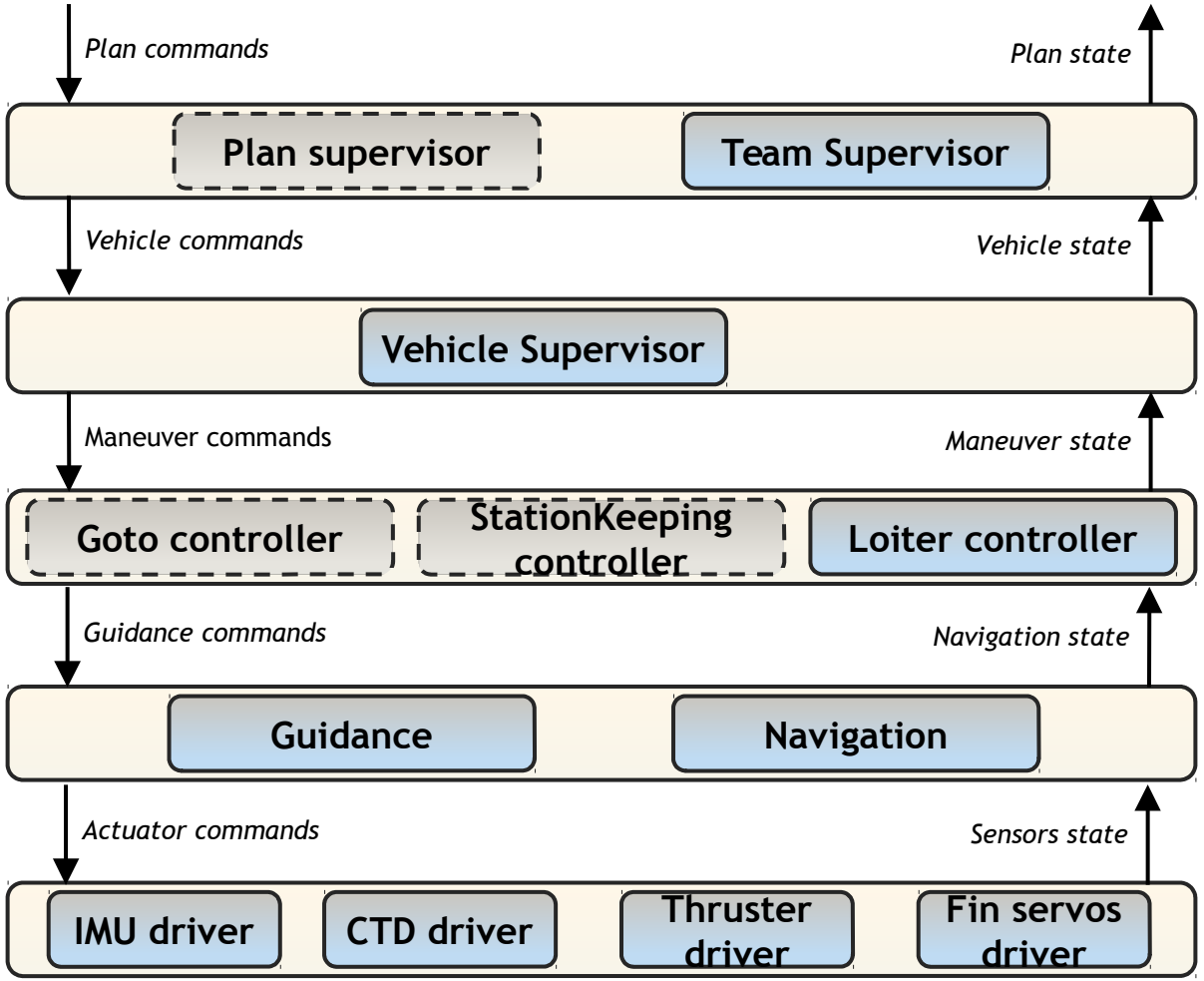
IMC: Inter-Module Communication API

Overview

- Message Oriented Protocol
- One XML document defines all messages
- Generators for documentation, C++ and Java code
- Serialization/deserialization to/from:
  - JSON
  - XML
  - **Binary**
- Serialized messages are used for logging and communication
- Binary serialization format can be translated to human-readable format (LLF)

Interaction Layers

- Plan control
- Vehicle control
- Maneuvering
- Guidance
- Navigation
- Sensing
- Actuation
- Networking
- Storage



Addressing

- Addresses are partitioned in classes (AUV, UAV, ROV, CCU, etc)
- Each system has a unique address (i.e., unique number)
- Subsystems/submodules of a system are called ***entities***
- Each entity has a unique local number used to further qualify a message (e.g., disambiguate messages of the same type but different sources, temperature from a CTD vs CPU temperature)

Anatomy of a message

- Synchronization Number
  - Marks the beginning of a message
  - Identifies protocol version
  - Allows for endianess detection
- Message Identification Number
  - Uniquely identifies a message type
- Message size
- Timestamp

Anatomy of a message

- Source Address
- Source Entity
- Destination Address
- Destination Entity
- ***Message Specific*** *Fields*
- CRC16

Example

```
<message id="263" name="Temperature" abbrev="Temperature">

  <description>
    Temperature measurement.
  </description>

  <field name="Value" abbrev="value" type="fp32_t" unit="°C">
    <description>
      Temperature value.
    </description>
  </field>

</message>
```

https://www.github.com/LSTS/imc

DUNE: Uniform Navigational Environment

Overview

- Designed for embedded systems
- Written in C++
- Used in AUVs, UAVs, ROVs, ASVs, data-loggers and communication gateways
- Related logical operations are isolated from each other in tasks, usually running in a separate thread of execution
- Communication between tasks and communication with external software is performed exclusively by using the set of messages described in the IMC API

Overview

- Communication
  - TCP, UDP, Acoustic modem, Iridium, GSM
- Logging
- Interaction with sensors, actuators, and power devices
- Controllers for attitude, speed, manual operation, etc
- Guidance algorithms
- Maneuvers (way-point following, area coverage, follow reference, loiter, station keeping, etc)

Supported Platforms

- Architectures
  - x86, ARM, PowerPC, SPARC, MIPS, AVR32
- Operating Systems
  - Linux v2.6+/Android, QNX v6.x, Oracle Solaris, Mac OS X, eCos, RTEMS, OpenBSD, FreeBSD, NetBSD, Microsoft Windows
- Hardware Interfaces
  - Serial Port, I²C, I/O port, CAN

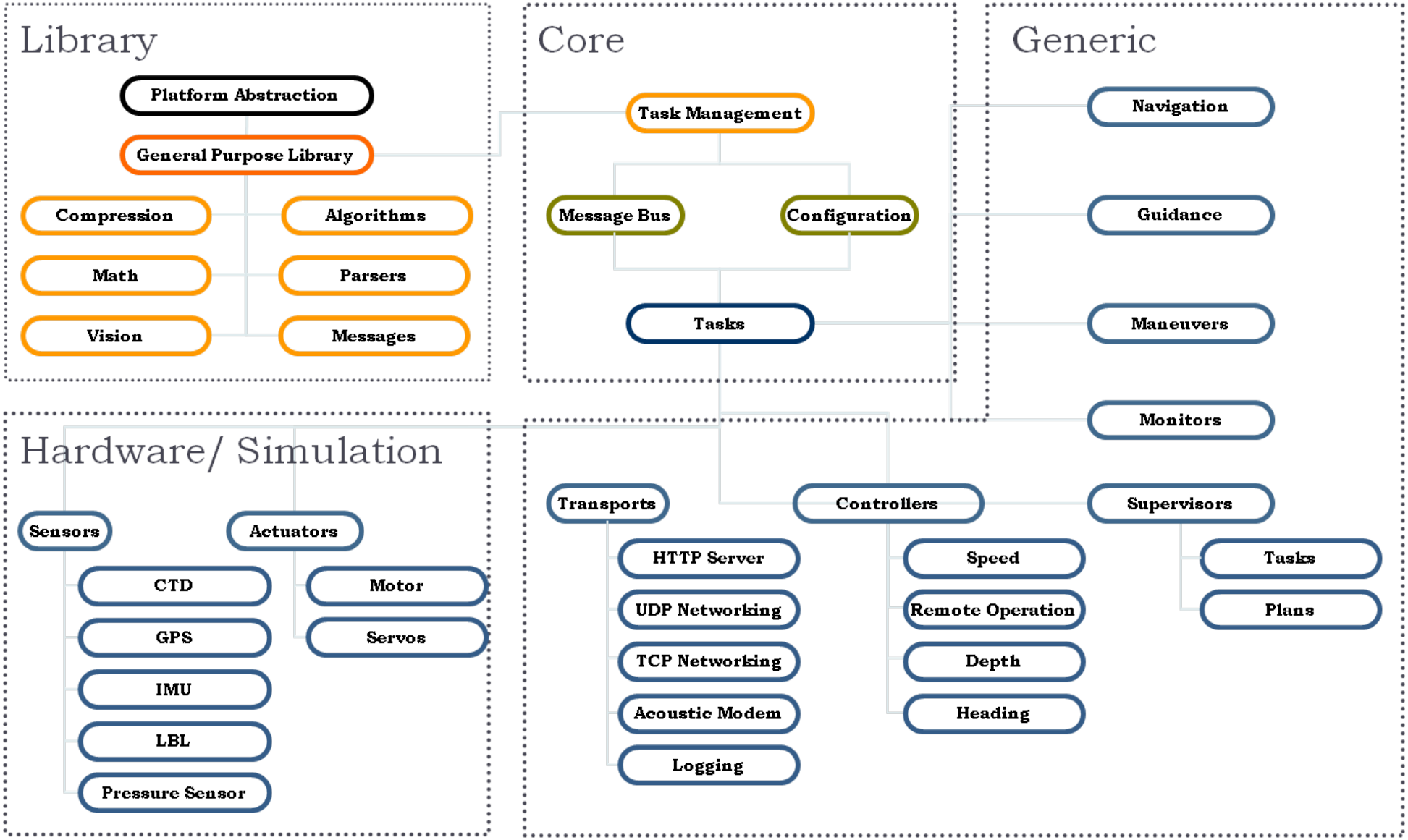
Required Software

- **Mandatory**
  - Git
  - CMake
  - C/C++ Compiler
  - Python Interpreter
- **Optional**
  - Eclipse
  - Microsoft Visual Studio

Required Software

- **Ubuntu/Debian**
  - sudo apt-get install cmake git g++ make python
- **Microsoft Windows**
  - http://www.cmake.org/download/
  - http://git-scm.com/downloads/
  - http://sf.net/projects/mingw/files/Installer/mingw-get-inst/
  - http://www.microsoft.com/express
- **Apple Mac OS X**
  - http://www.cmake.org/download/
  - https://developer.apple.com/xcode/

Example System



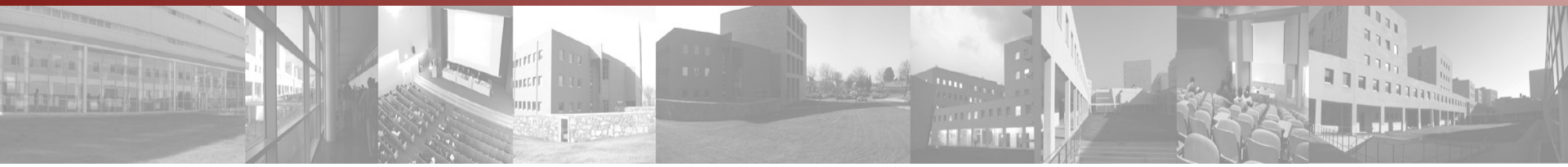
Resources

- **Source Code**
  - https://github.com/LSTS/dune
- **Documentation**
  - http://lsts.pt/docs
  - https://github.com/LSTS/dune/wiki
- **Mailing List**
  - https://groups.google.com/forum/#!forum/lsts-toolchain
  - lsts-toolchain@googlegroups.com
- **Nightly Builds**
  - http://www.lsts.pt/cdash/index.php?project=DUNE

Nightly Builds

Nightly

Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
macosx-8-x86-64	x86-32bit-darwin-apple-clang	0	0	0	0_{-50}	0	0_{-13}	0	13^{+13}	Nov 13, 2014 - 07:55 GMT
dragonflybsd-3-x86-32	x86-32bit-dragonfly-bsd-gcc4x	0	0	0	0	0	0	0	13	Nov 13, 2014 - 10:31 GMT
freebsd-10-x86-64	x86-32bit-freebsd-libcxx-clang	0	0	0	0_{-50}	0_{-50}	0_{-2}	0	13^{+2}	Nov 13, 2014 - 13:33 GMT
he162	x86-32bit-linux-glibc-clang	0	0	0	0_{-50}	0	0_{-13}	0	13^{+13}	Nov 13, 2014 - 14:29 GMT
ubuntu-12-x86-64	x86-32bit-linux-glibc-clang	0	0	0	0_{-50}	0	0_{-13}	0	13^{+13}	Nov 13, 2014 - 03:09 GMT
ubuntu-12-x86-32	x86-32bit-linux-glibc-clang	0	0	0	0_{-50}	0	0_{-13}	0	13^{+13}	Nov 13, 2014 - 04:02 GMT
ubuntu-13-x86-64	x86-32bit-linux-glibc-clang	0	0	0	0_{-50}	0	0_{-13}	0	13^{+13}	Nov 13, 2014 - 13:07 GMT
he162	x86-32bit-linux-glibc-gcc4x	0	0	0	0	0	0	0	13	Nov 13, 2014 - 14:02 GMT
centos-6-x86-64	x86-32bit-linux-glibc-gcc4x	0	0	0	0	1	0	0	13	Nov 13, 2014 - 01:28 GMT
ubuntu-12-x86-64	x86-32bit-linux-glibc-gcc4x	0	0	0	0	0	0	0	13	Nov 13, 2014 - 02:25 GMT
ubuntu-12-x86-32	x86-32bit-linux-glibc-gcc4x	0	0	0	0	0	0	0	13	Nov 13, 2014 - 03:41 GMT
debian-6-x86-64	x86-32bit-linux-glibc-gcc4x	0	0	0	0	1	0	0	13	Nov 13, 2014 - 08:36 GMT



Neptus – Command and Control Infrastructure

What's Neptus?

- Neptus allows planning, control and revision of missions performed by unmanned vehicles
- Neptus supports multiple heterogeneous vehicles
  - AUVs, UAVs, ROVs, ASVs, ...
  - Controlled individually or as a team
- Neptus supports multiple operators
  - Operators join in and access / control the network of vehicles
- Neptus can be extended through plug-ins
  - Map layers, Data visualizations, Console widgets, Maneuvers, Communication protocols, ...

Neptus mission concept

- In Neptus, a mission is specified as
  - A set of map features
  - A set of programmed plans
  - A set of vehicle configurations

- The mission is usually...
  - Created prior to execution (planning)
  - Changed during execution (monitoring / revision / re-planning)

Part 1: Using Neptus

Neptus Requirements

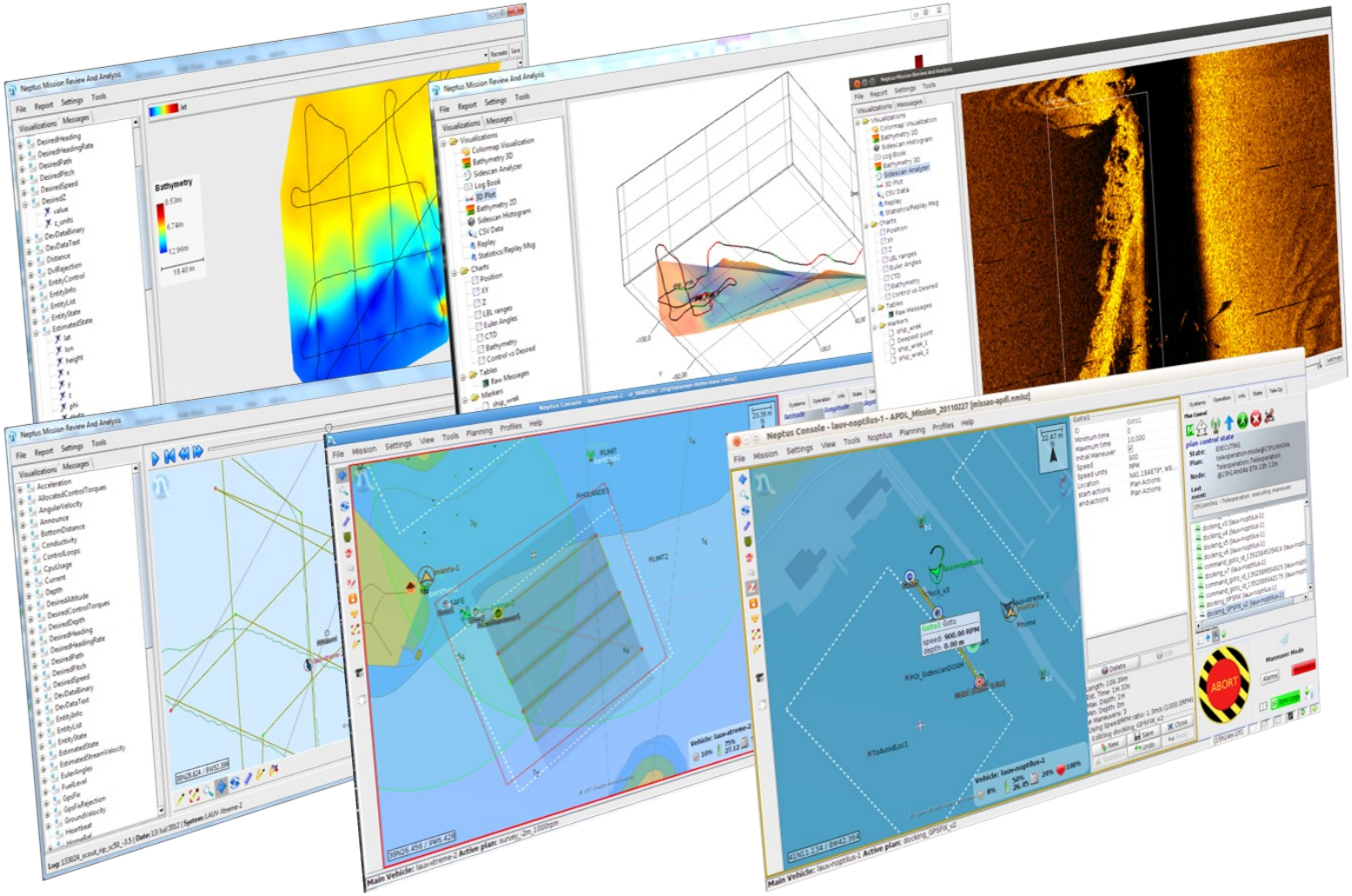
- Neptus requires prior installation of Oracle's Java Runtime Environment version 8
- For 3D widgets an OpenGL-compatible graphics adapter is recommended
  - For some 3D data visualizations, VTKJava library is required
- At least 1 GB of RAM (4 GB recommended)
- Compatible with Windows and Linux (known to work under OSX but rarely tested)

Installing and Running Neptus

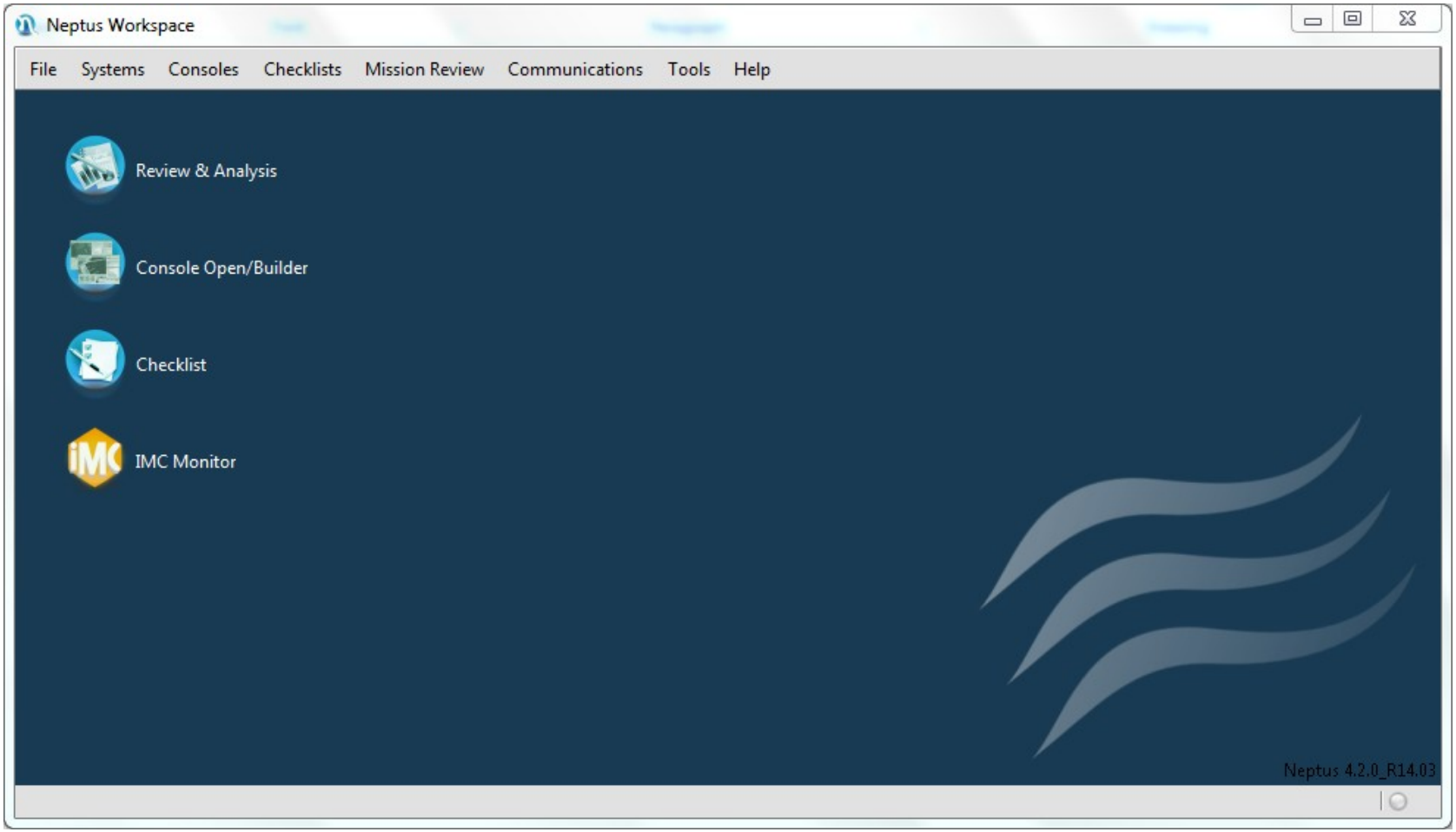
- To install Neptus, just download the latest version to a directory of choice
  - Logs will be put under this directory so make sure you leave extra disk space for them
- Downloading Neptus
  - Use your favorite Git client to clone Neptus from https://github.com/LSTS/neptus

    git glone https://github.com/LSTS/neptus.git
- Running Neptus
  - In Windows: run neptus.bat
  - In Linux: execute ./neptus.sh

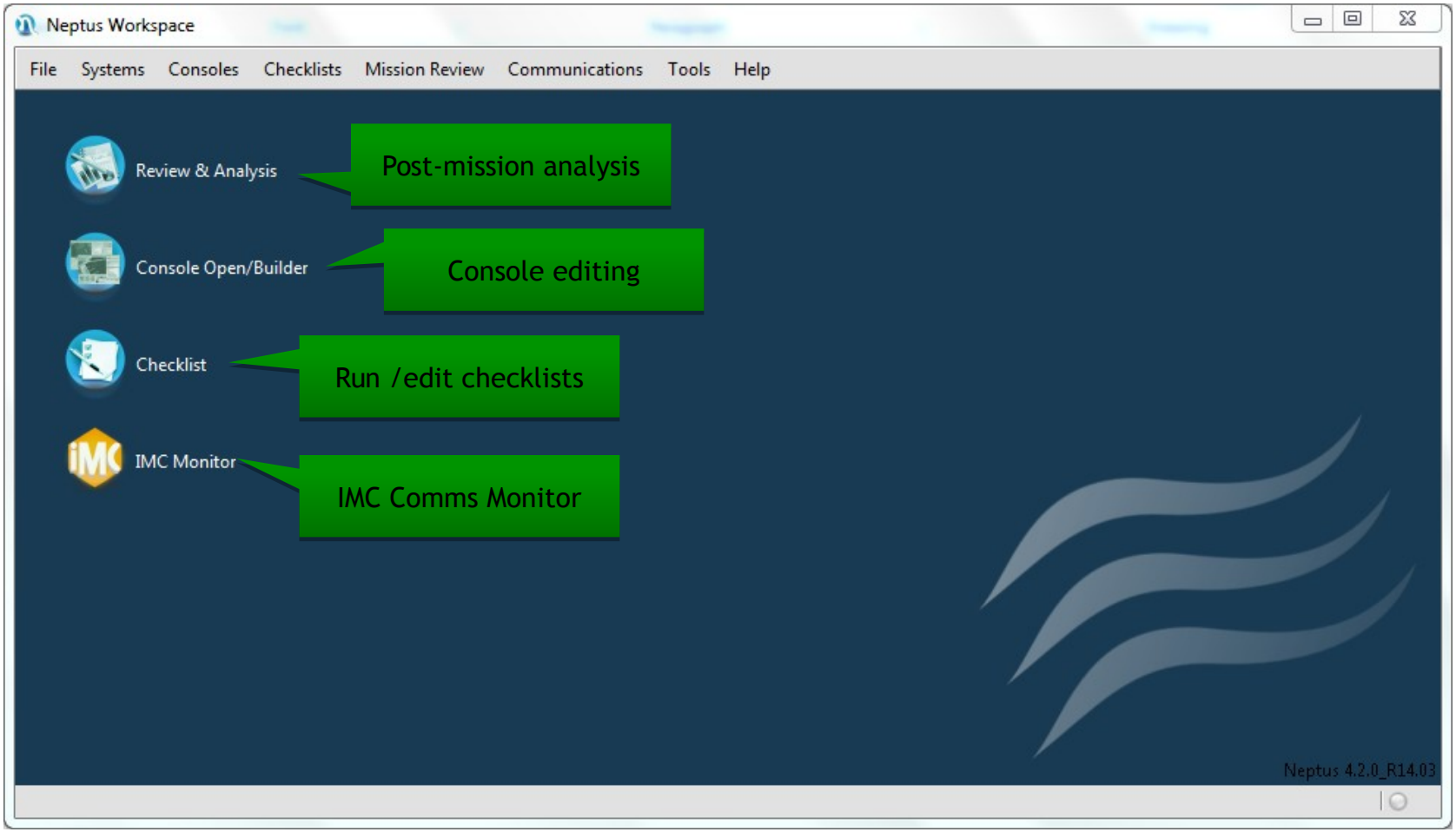
Interfaces Adjusted/Adjustable to Several Needs



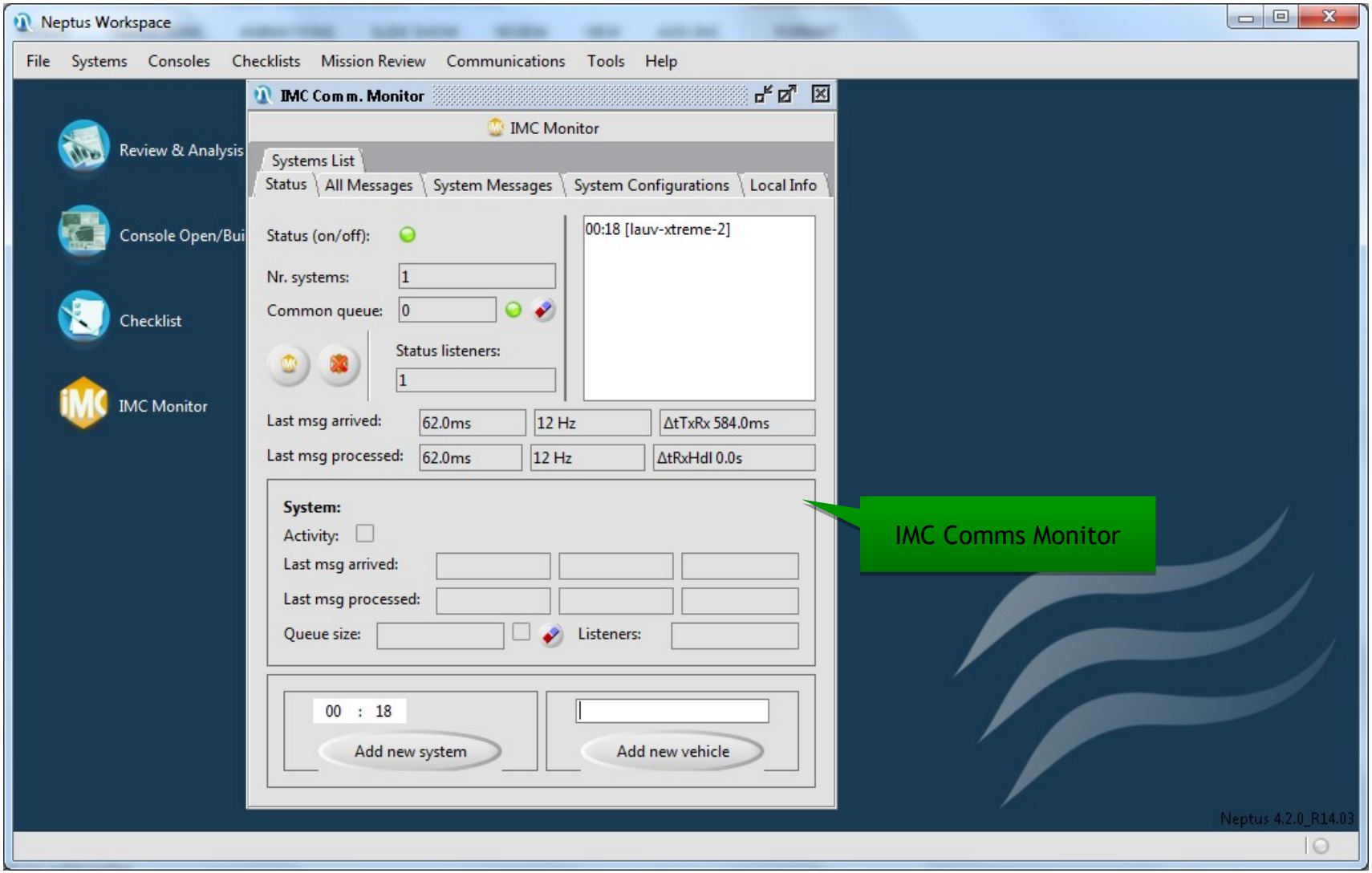
The Neptus Workspace



The Neptus Workspace



The Neptus Workspace

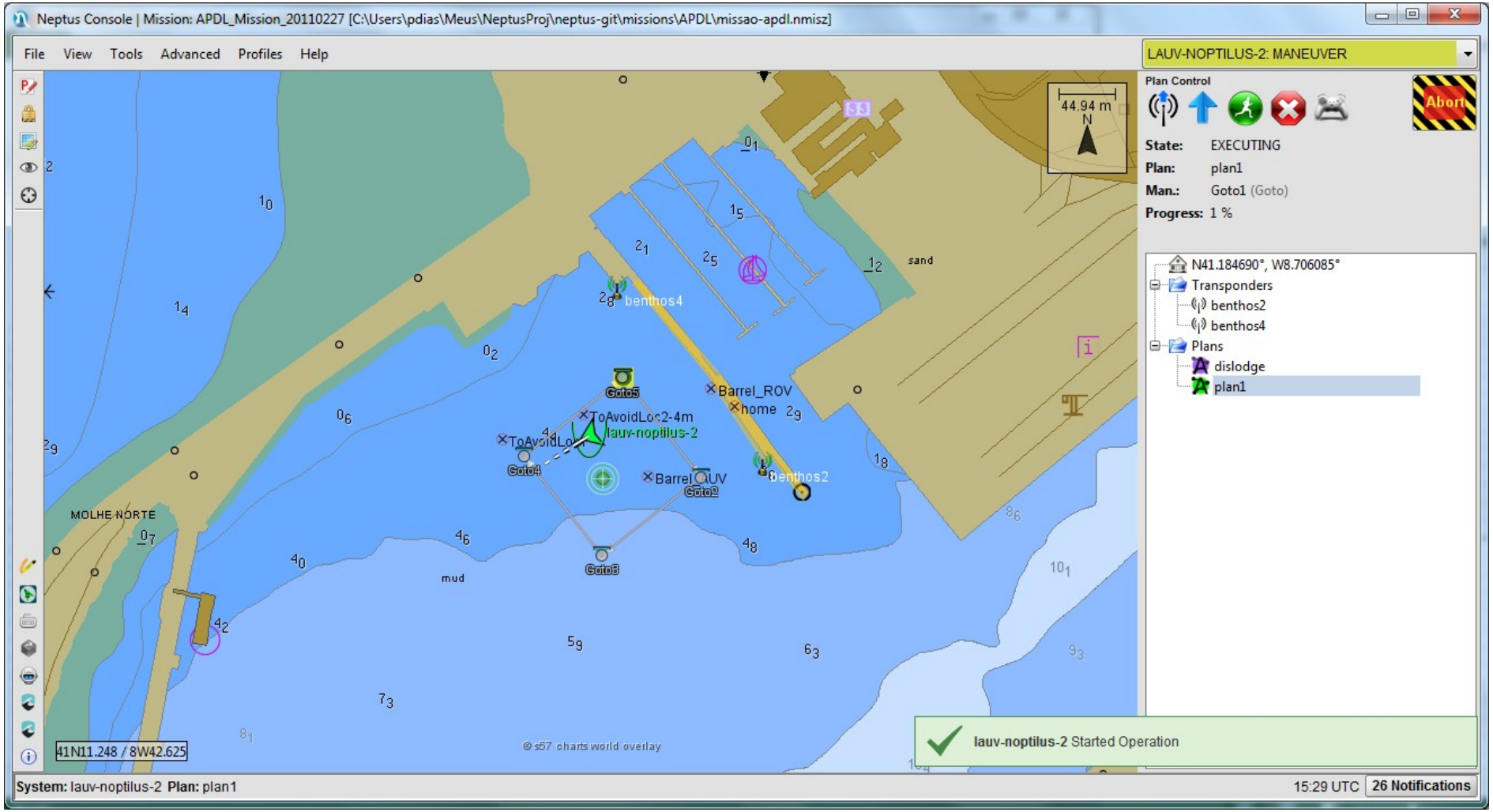


Neptus Consoles

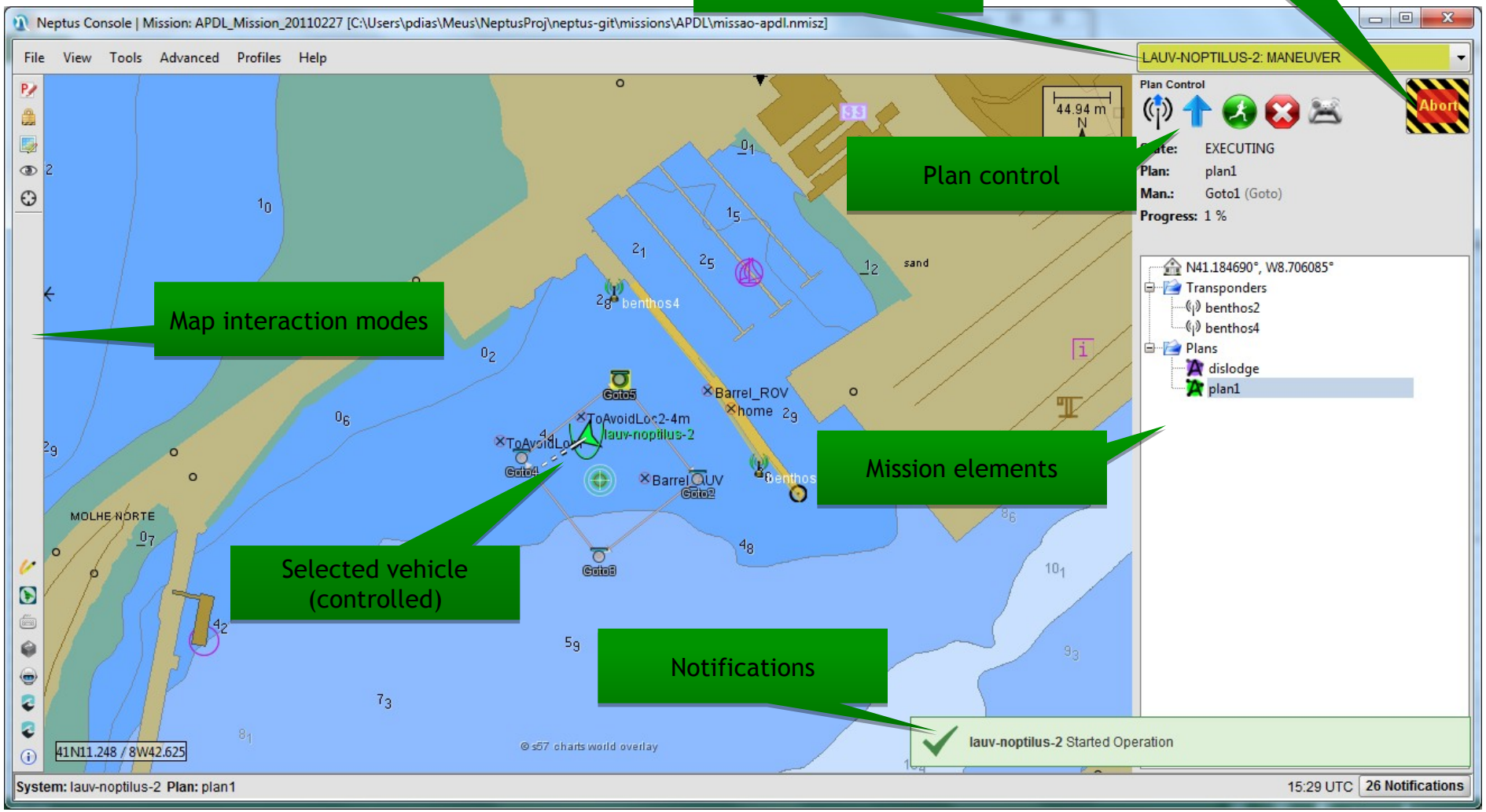
- Neptus allow end-users to create Operational Consoles
  - Based on existing widgets
  - Adapted to specific missions/vehicles

- Mission console definitions are stored as XML
  - .ncon file extension
  - A sort of consoles are already bundled

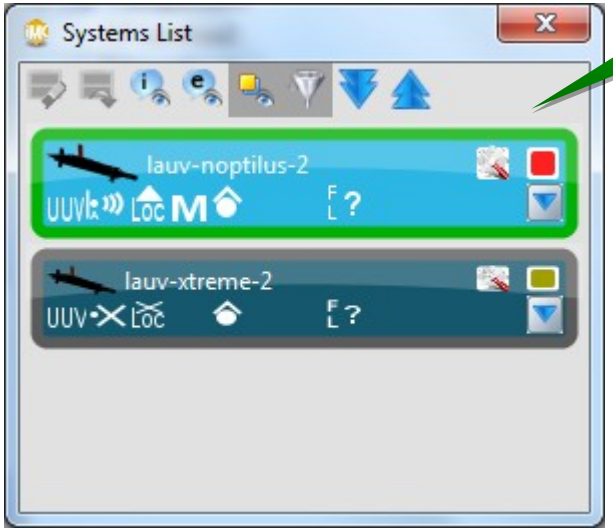
Neptus Consoles



Neptus Consoles

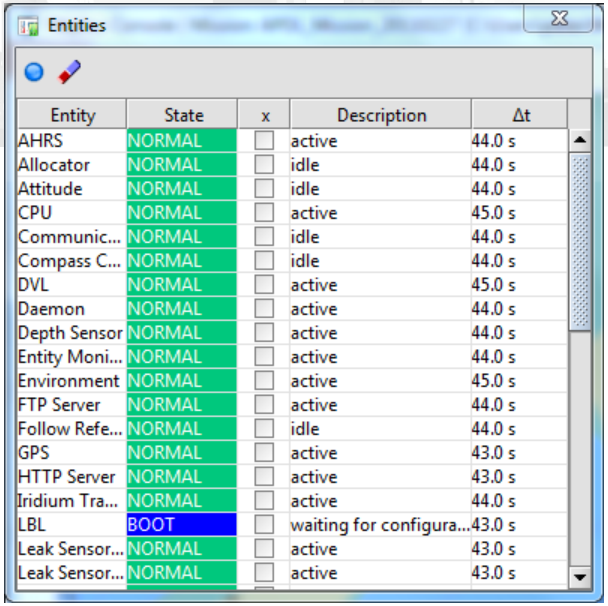


Neptus Consoles

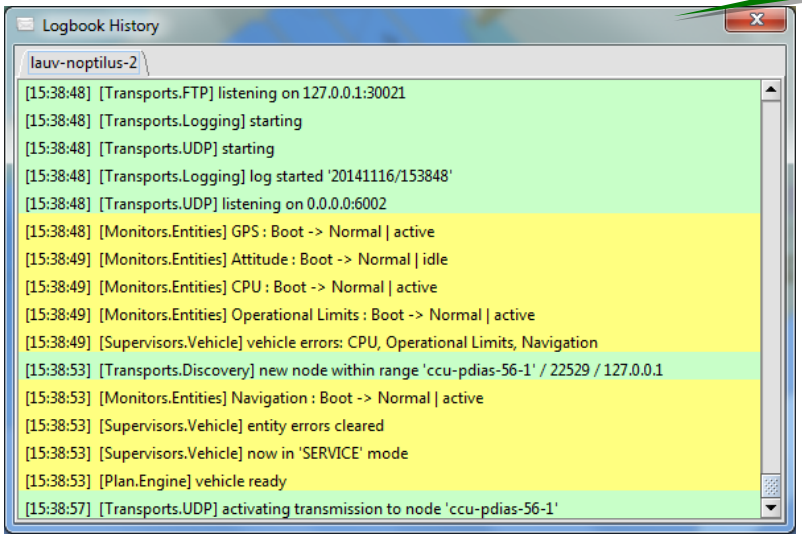


Systems listing and selection

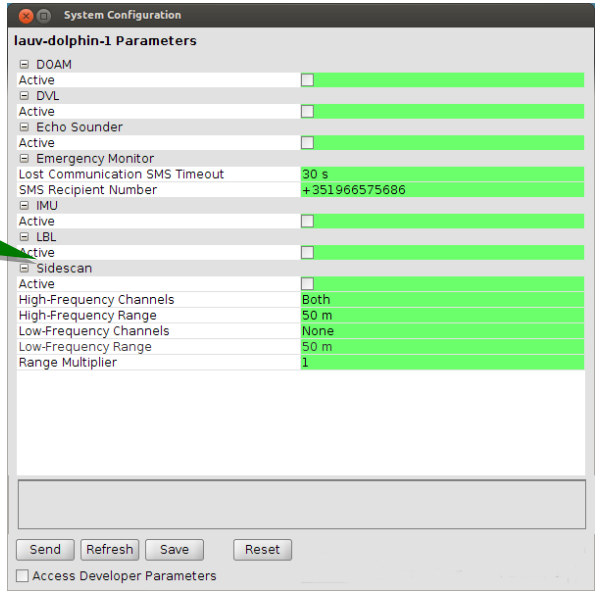
Vehicle subsystems state



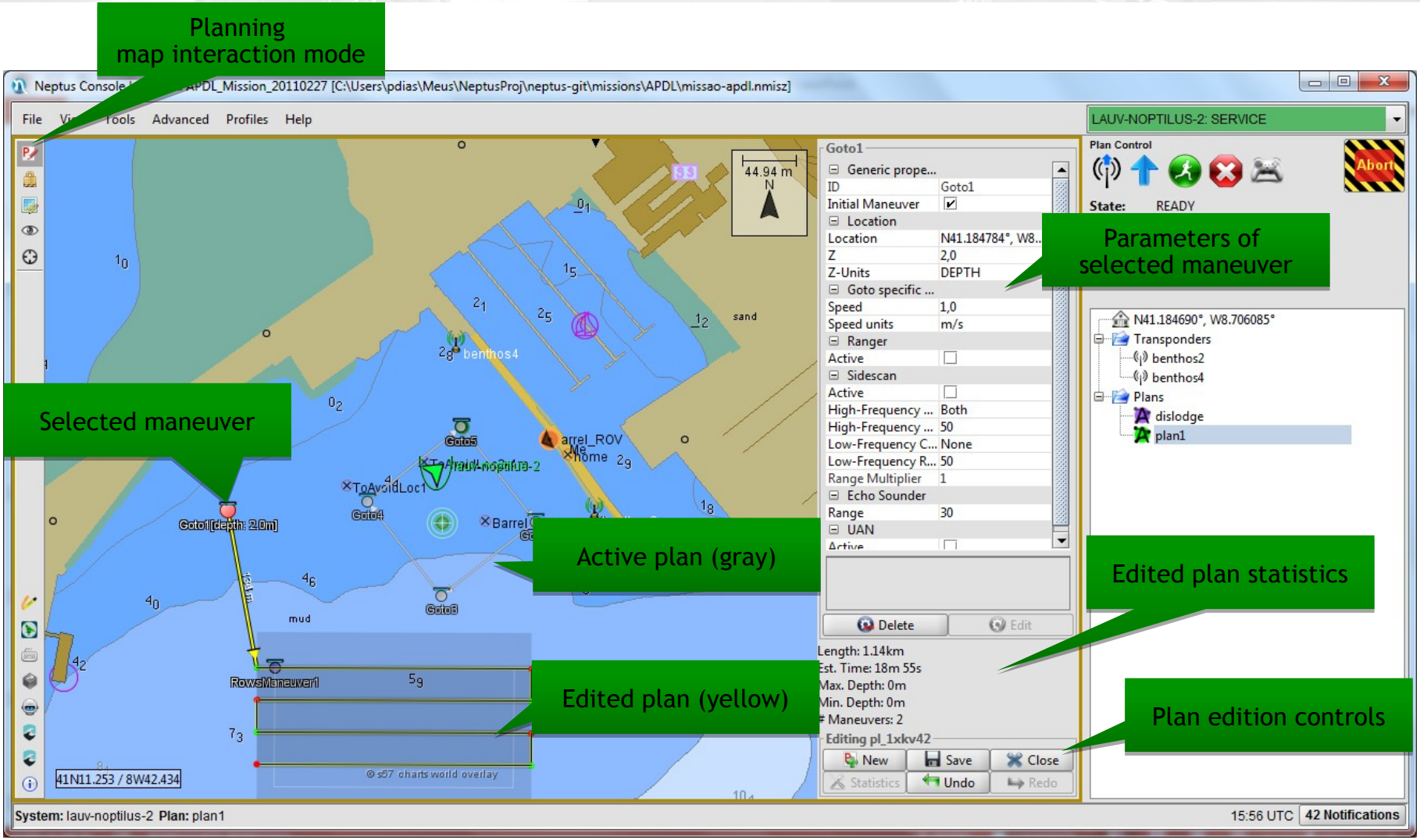
Vehicle Log Book



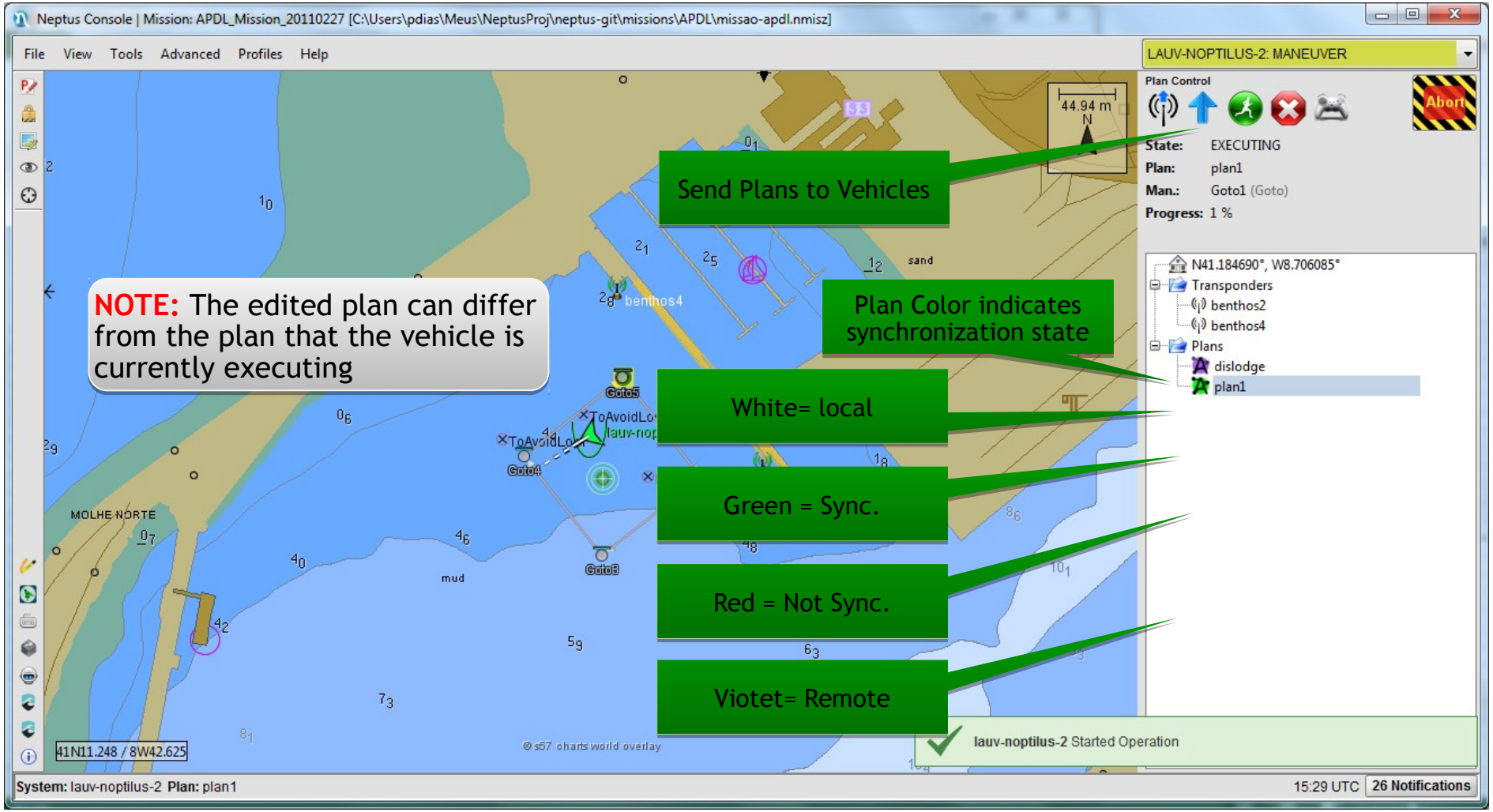
Vehicles Configurations



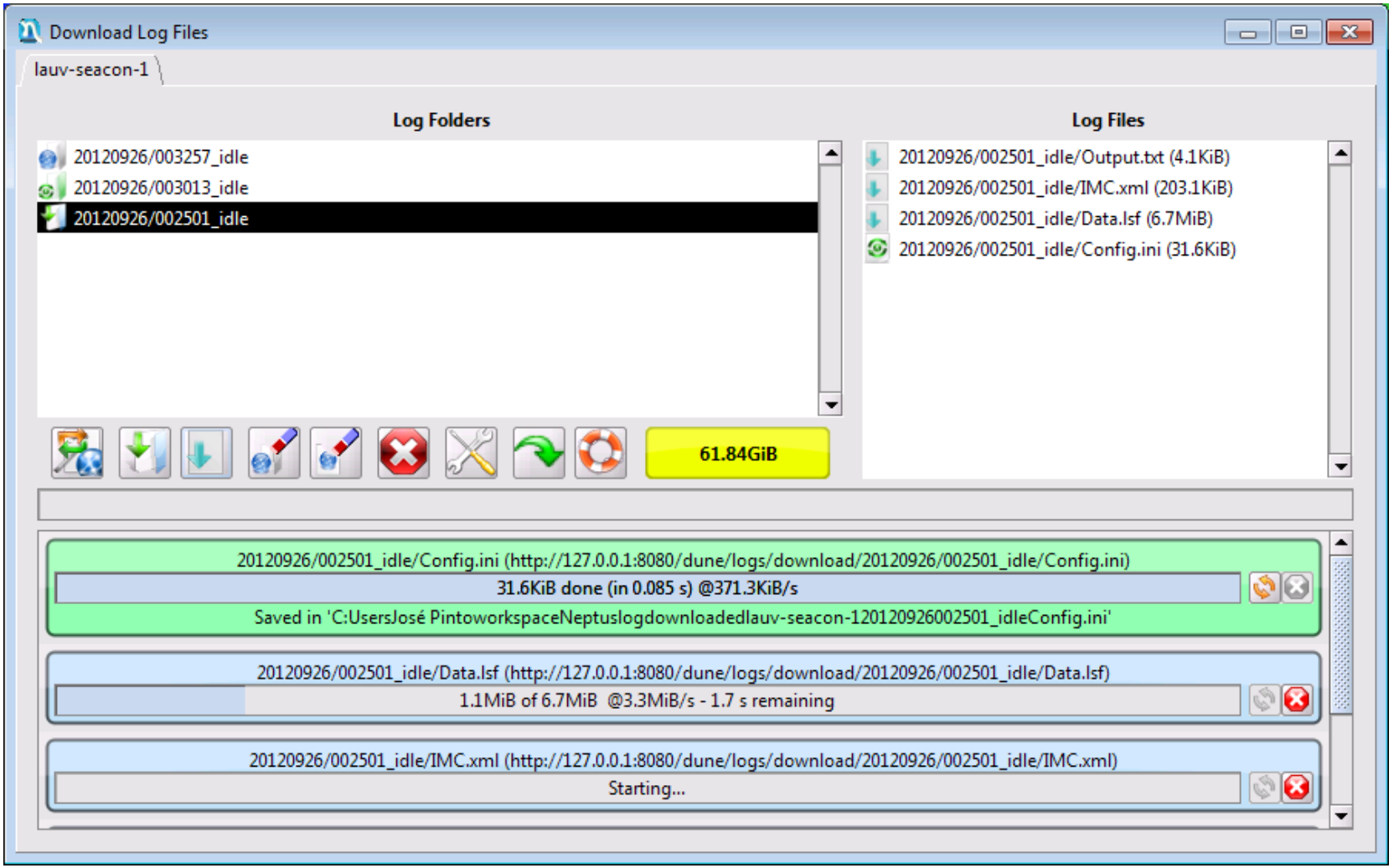
Neptus Consoles



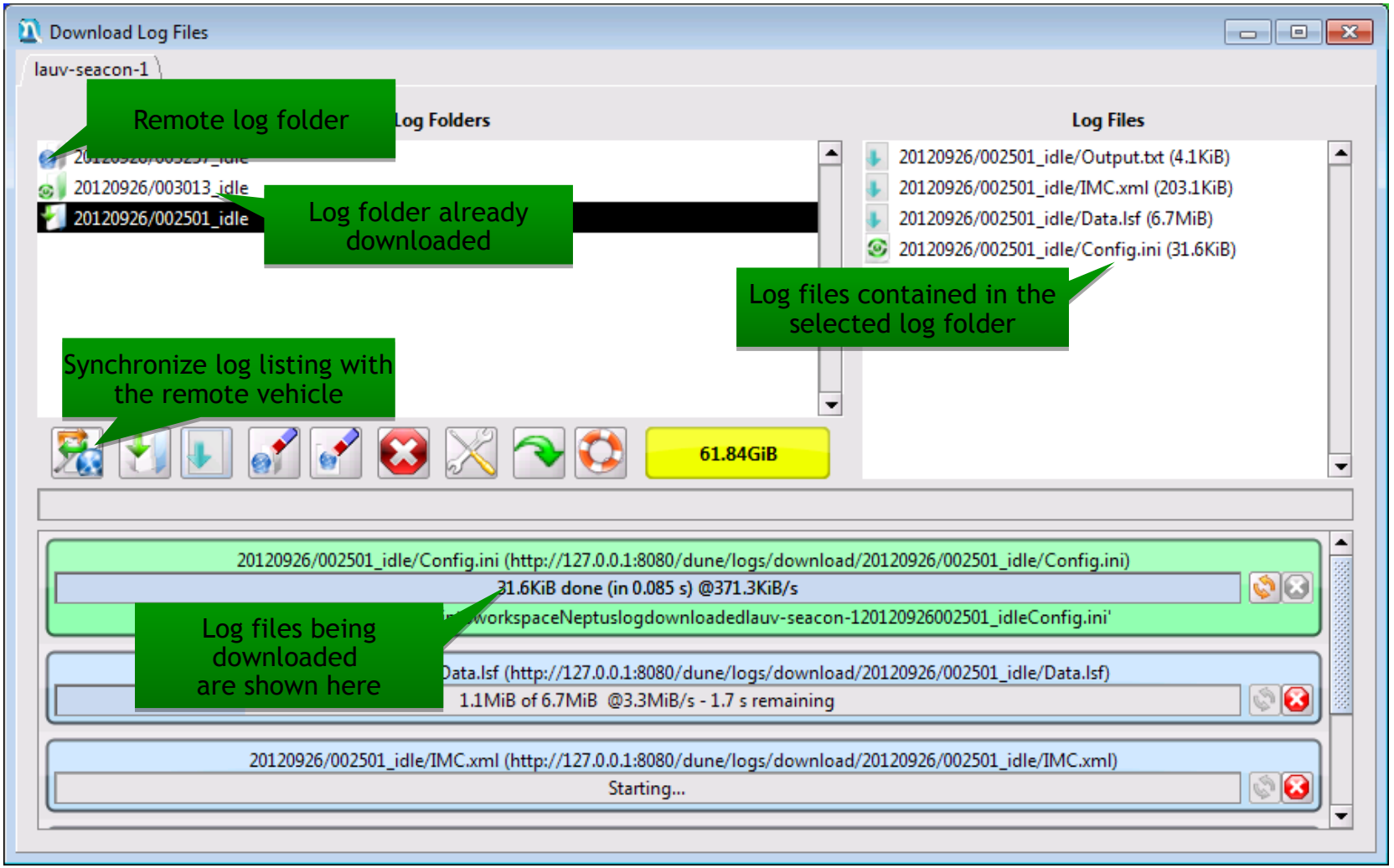
Neptus Consoles



Neptus Consoles - Log Download Dialog



Neptus Consoles - Log Download Dialog

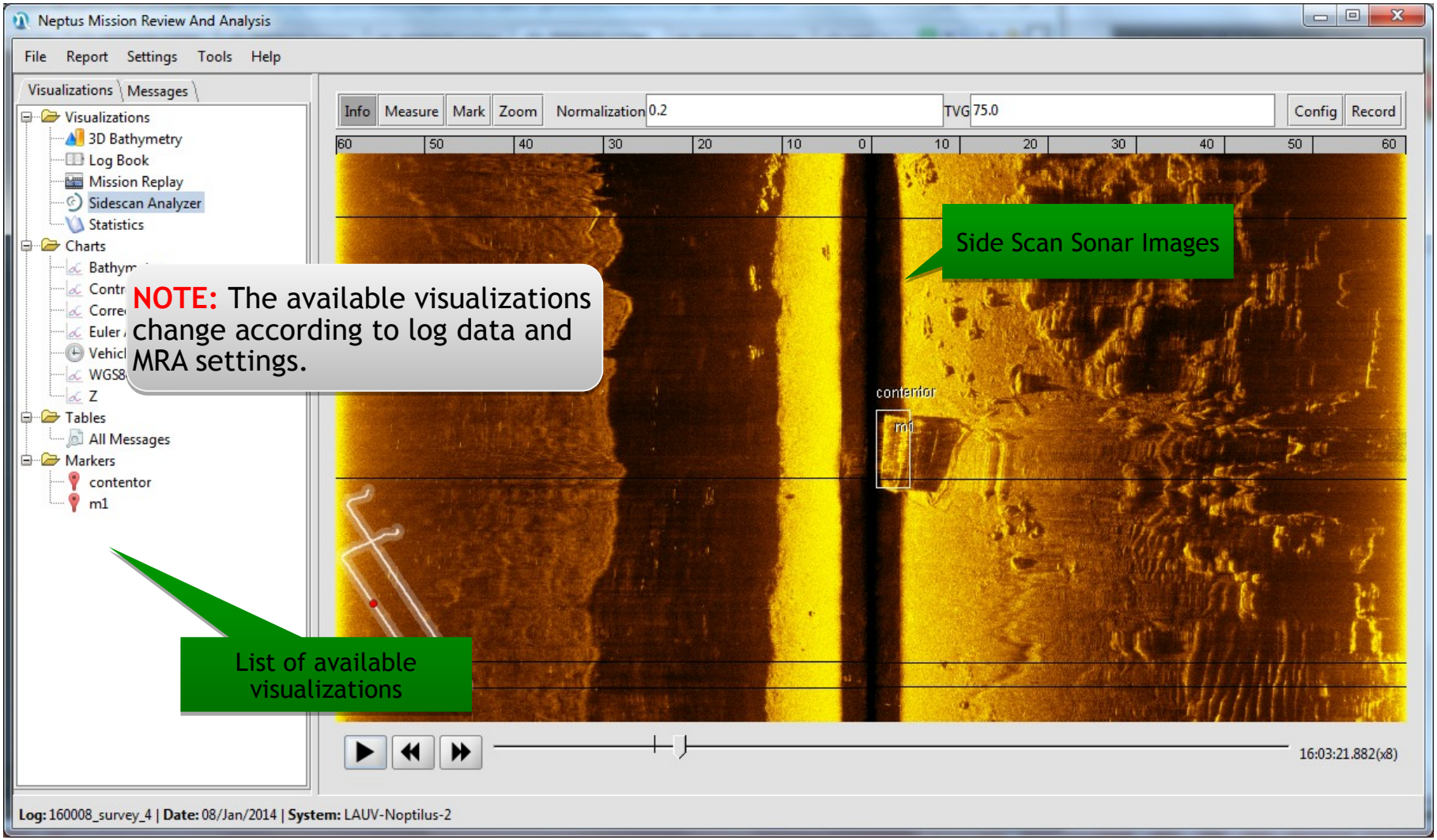


Neptus Mission Review and Analysis

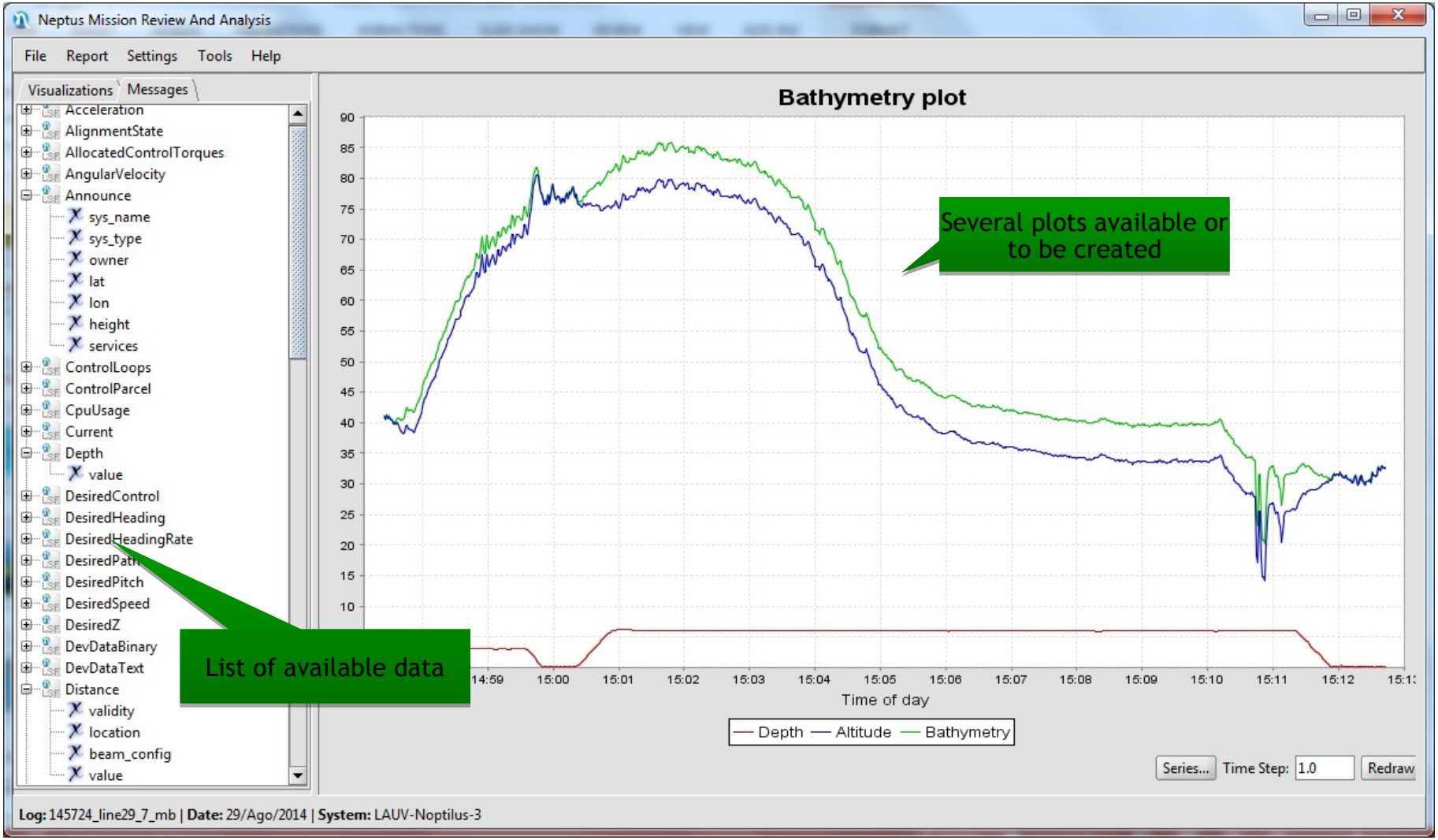
- Can be acessed
  - Directly by right-clicking a downloaded log
  - From the Neptus workspace

- Compatible with LSF log folders
  - Data.lsf (binary concatenation of IMC data)
  - IMC.xml (definition of the protocol used in the LSF)
  - config.ini (used vehicle configuration)

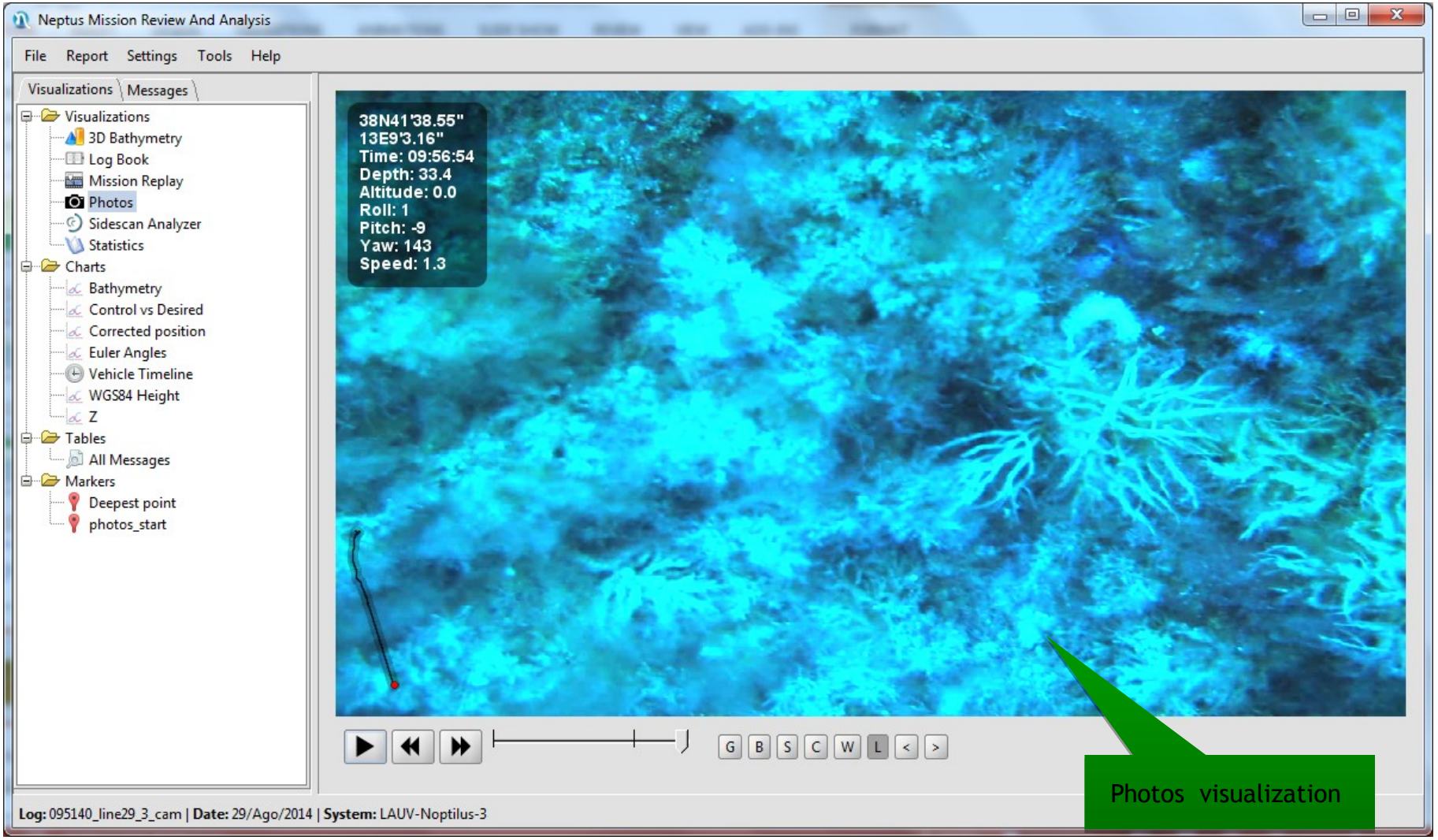
Neptus Mission Review and Analysis



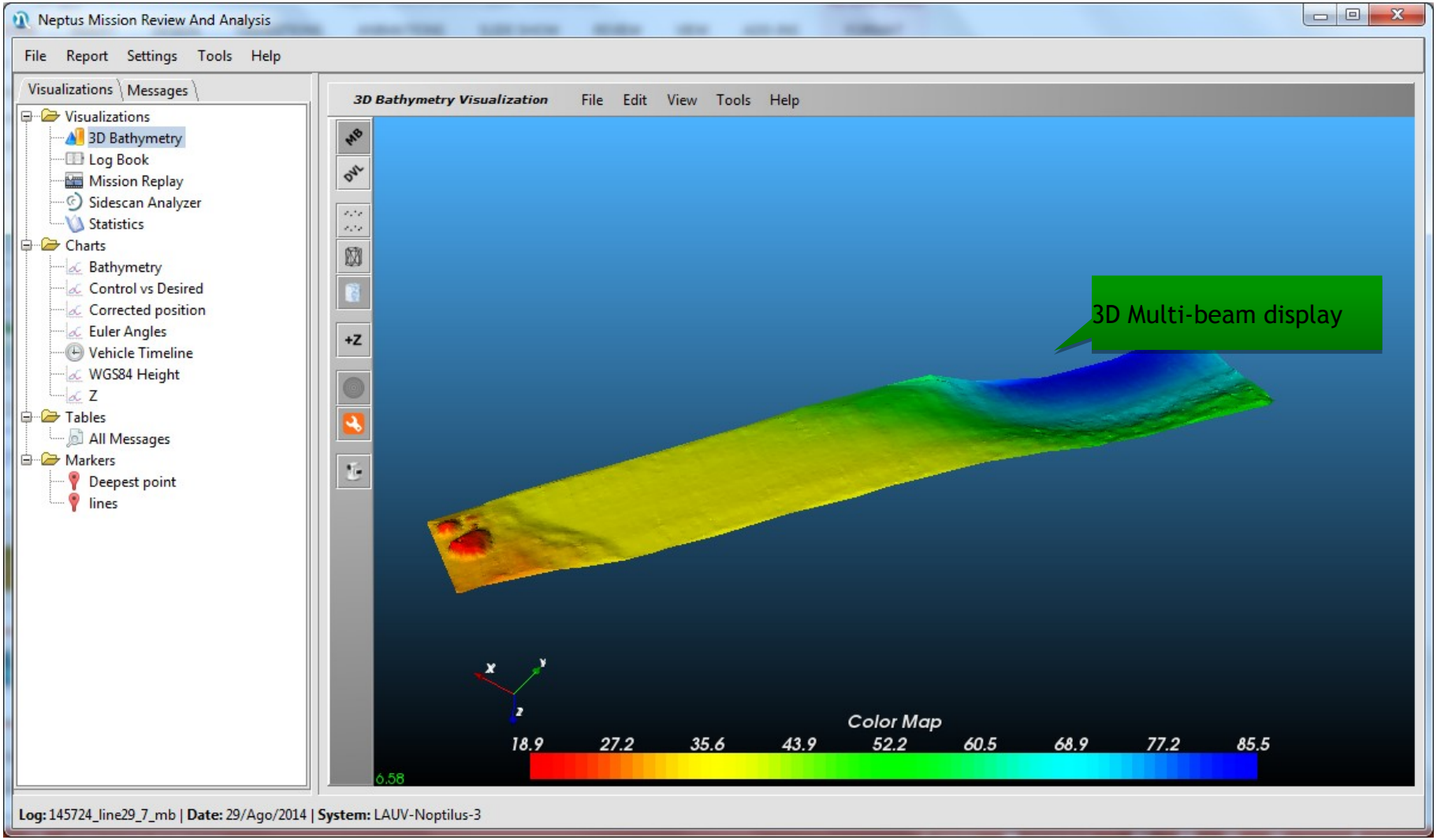
Neptus Mission Review and Analysis



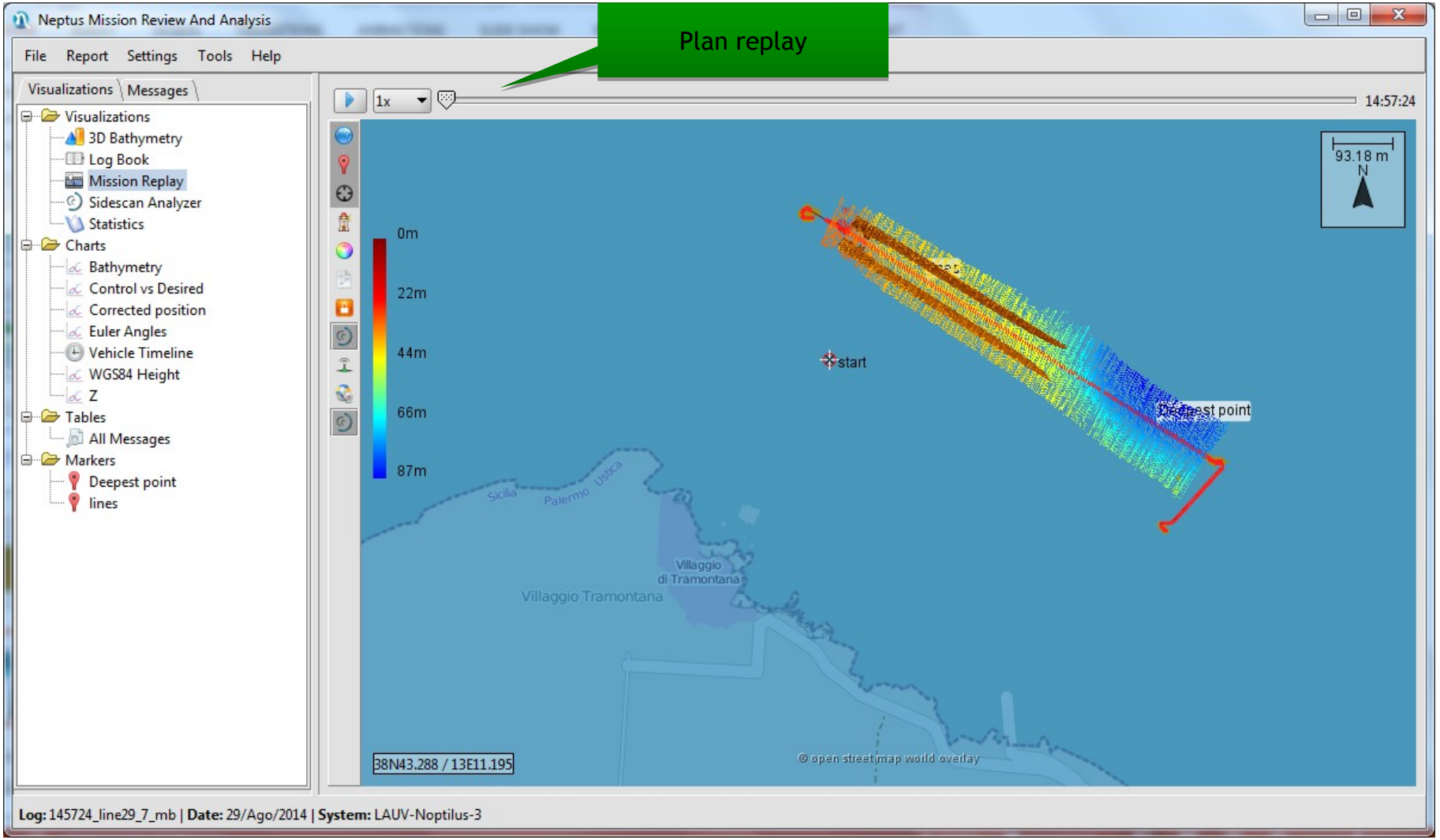
Neptus Mission Review and Analysis



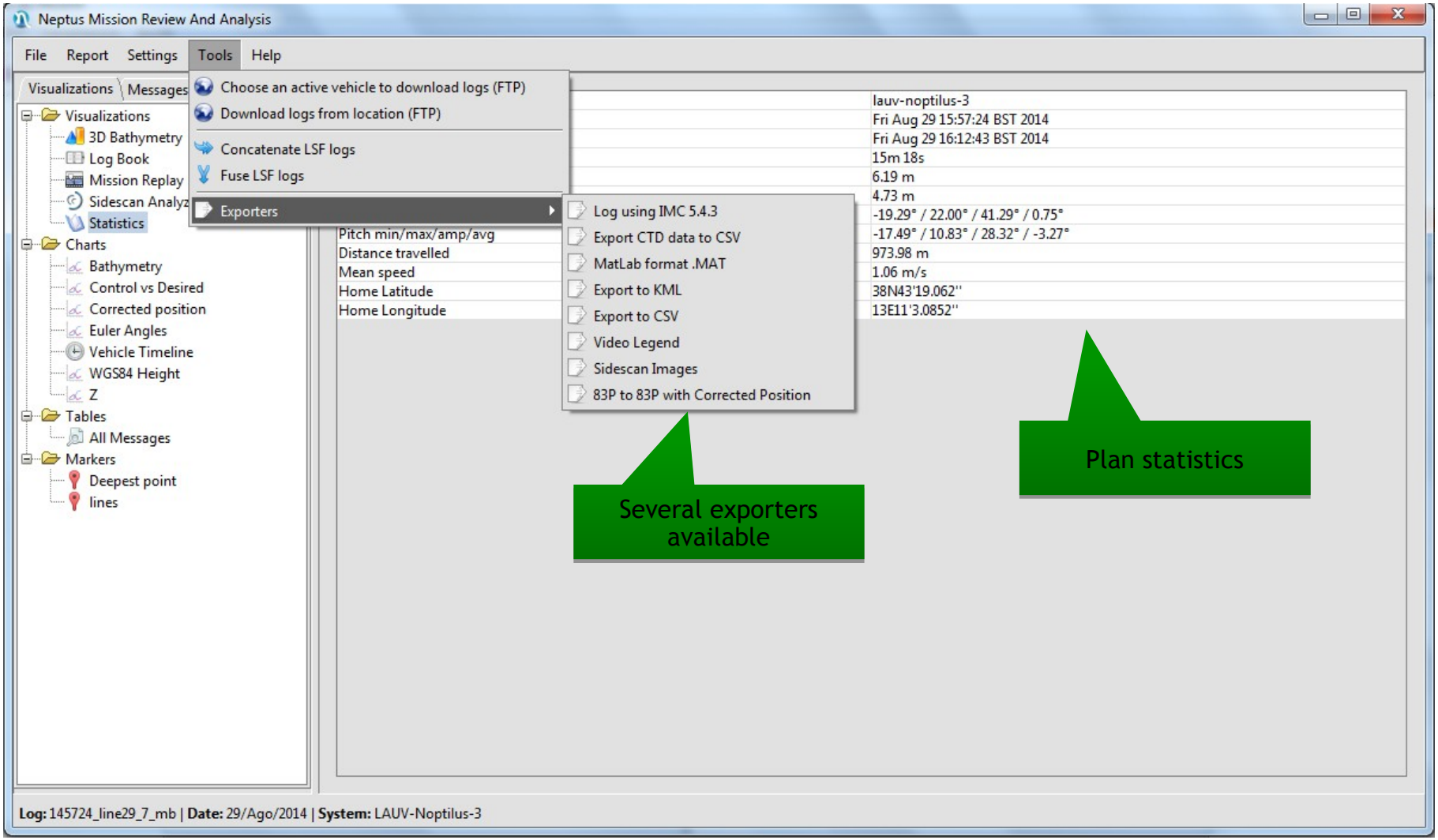
Neptus Mission Review and Analysis



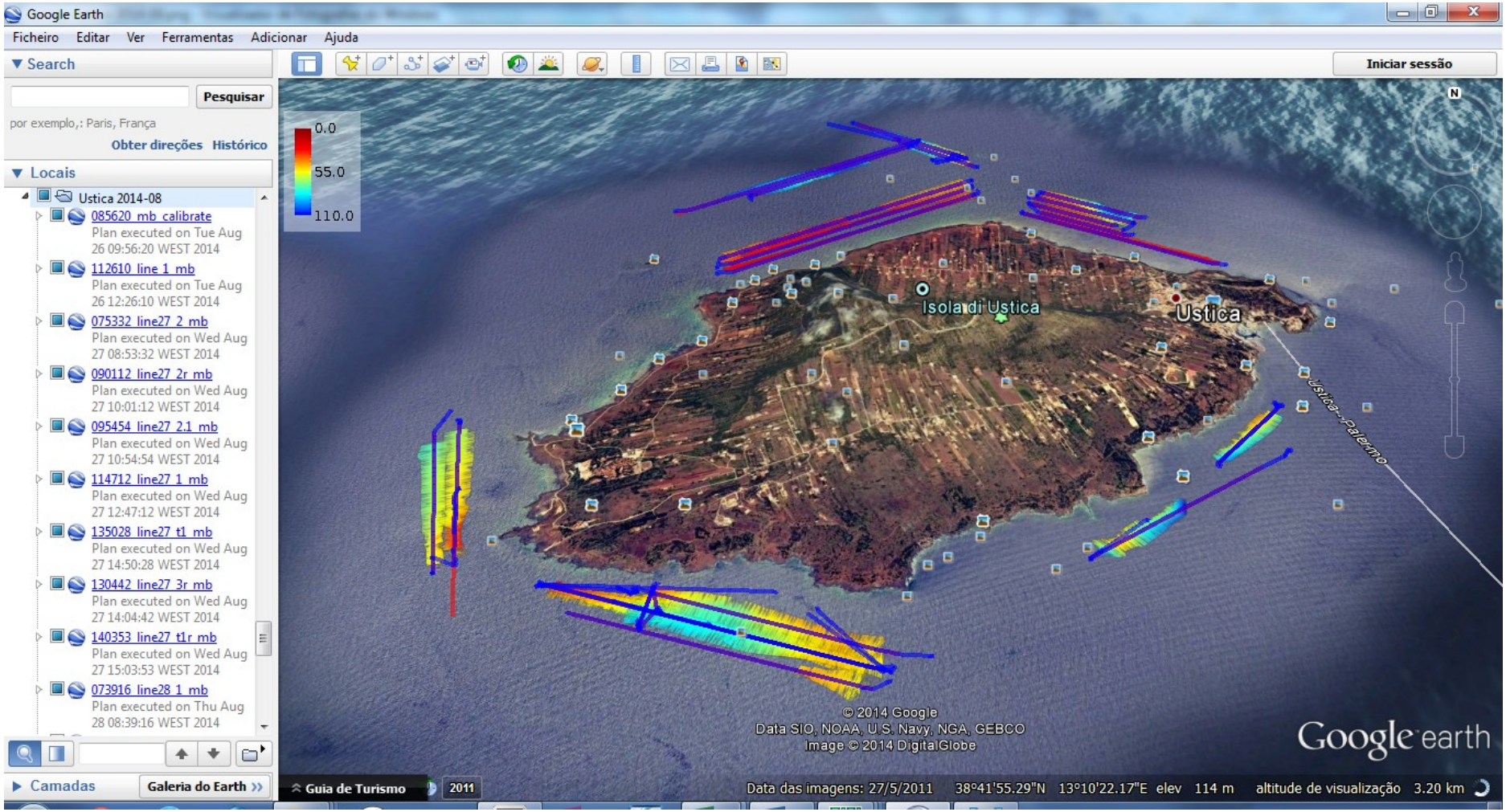
Neptus Mission Review and Analysis



Neptus Mission Review and Analysis



Neptus KML Export



Extending DUNE

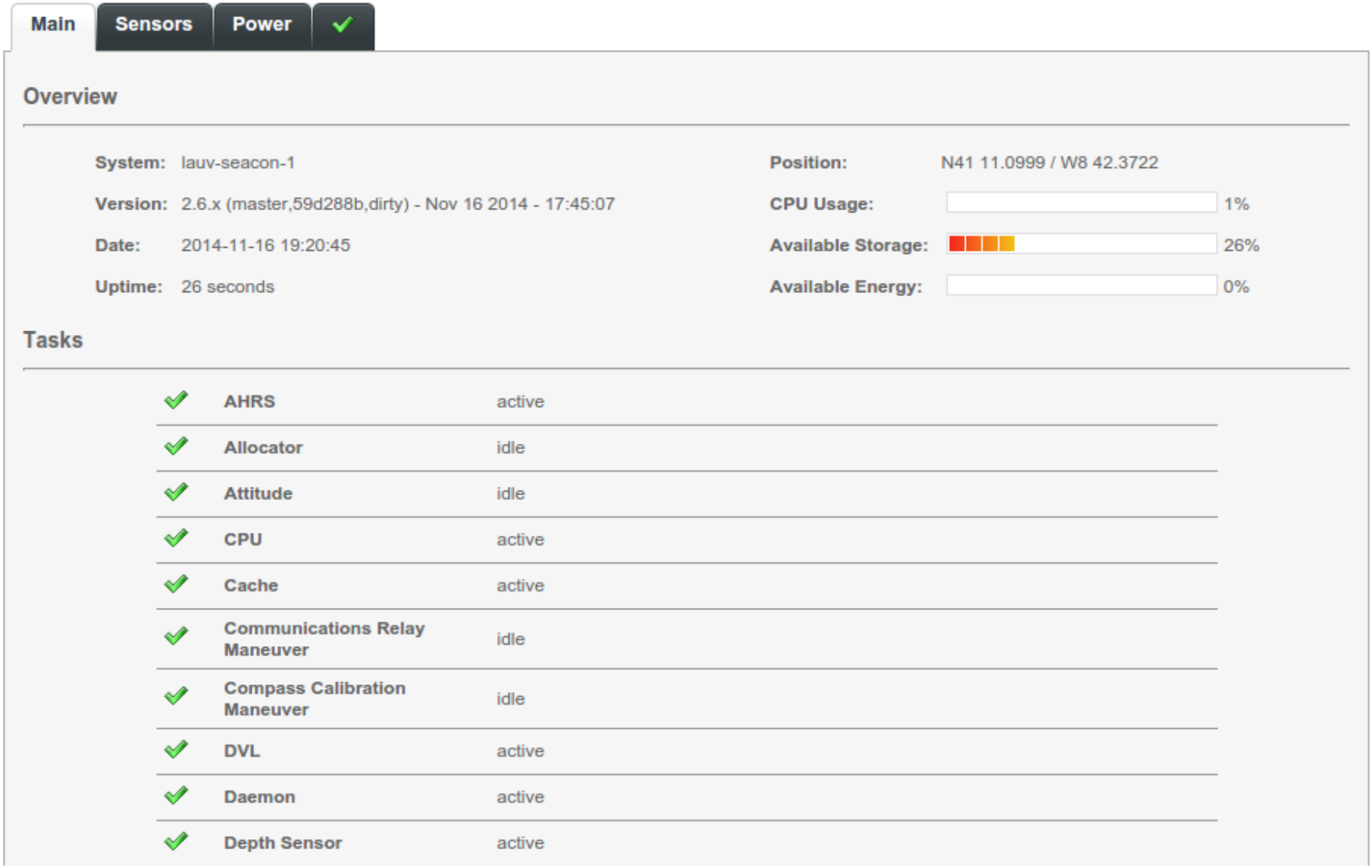
Compilation and Execution

- mkdir $HOME/tutorial && cd $HOME/tutorial
- git clone https://github.com/LSTS/dune.git dune
- mkdir build && cd build
- cmake ../dune -G "Eclipse CDT4 - Unix Makefiles"
- make
- ./dune -c lauv-seacon-1 -p Simulation

  - Optional: have a look in **dune/src/etc/lauv-seacon-1.ini**

Web Interface

- http://127.0.0.1:8080



Anatomy of a Task

- Runs concurrently with other tasks
- Communicates with other tasks using IMC messages
- Does one job (and does it right)
- Can be event-driven or periodic

Basic Functions

- **Task**(const std::string& name, Tasks::Context& ctx)
  - Task constructor
  - Never fails, doesn't throw exceptions
  - Declares configuration parameters
  - Allocates resources that do not depend on configuration parameters
- void **onUpdateParameters**(void)
  - Called when configuration parameters change
- void **onEntityReservation**(void)
  - Called when the task can reserve entities

Example Task: Consumer

- http://goo.gl/1n2Cpk
- Task consumes temperature messages and prints them to the output (console)
- Scaffold created using the command:
  - python ../dune/programs/scripts/dune-create-task.py ../dune 'Ricardo Martins' 'Workshop/Consumer'
  - make rebuild_cache
- Source code resides in ***src/Workshop/Consumer***
- Task entry point is ***src/Workshop/Consumer/Task.cpp***
- Build: make

Example Task: Consumer

```
 1  // DUNE headers.
 2  #include <DUNE/DUNE.hpp>
 3
 4  namespace Workshop
 5  {
 6    //! Simple task that consumes temperature messages and prints them to
 7    //! the terminal.
 8    namespace Consumer
 9    {
10      using DUNE_NAMESPACES;
11
12      //! Entry point.
13      struct Task: public Tasks::Task
14      {
```

Example Task: Consumer

```
15        //! Task constructor.
16        Task(const std::string& name, Tasks::Context& ctx):
17          Tasks::Task(name, ctx)
18        {
19          bind<IMC::Temperature>(this);
20        }
21
22        //! Process temperature messages.
23        void
24        consume(const IMC::Temperature* msg)
25        {
26          inf("temperature is %f", msg->value);
27        }
```

Example Task: Consumer

```
28
29          //! Main loop.
30          void
31          onMain(void)
32          {
33            while (!stopping())
34            {
35              waitForMessages(1.0);
36            }
37          }
38        };
39      }
40    }
41
42    DUNE_TASK
43
```

Basic Functions

- void **onEntityResolution**(void)
  - Called when the task can resolve entities
- void **onResourceAcquisition**(void)
  - Called when the task can acquire resources (open serial ports, sockets, etc)
- void **onResourceInitialization**(void)
  - Called when the task can initialize previously acquired resources
- void **onResourceRelease**(void)
  - Releases all acquired resources
- void **onMain**(void) / void **task**(void)
  - Main task loop

Example Task: Producer

- http://goo.gl/FUezwX
- Task produces random temperature values and dispatches them to the message bus
- Scaffold created using the command:
  - python ../dune/programs/scripts/dune-create-task.py ../dune 'Ricardo Martins' 'Workshop/Producer'
  - make rebuild_cache
- Source code resides in ***src/Workshop/Producer***
- Task entry point is ***src/Workshop/Producer/Task.cpp***
- Build: make

Example Task: Producer

```
1  // DUNE headers.
2  #include <DUNE/DUNE.hpp>
3
4  namespace Workshop
5  {
6    //! Simple task that produces random temperature measurements.
7    namespace Producer
8    {
9      using DUNE_NAMESPACES;
```

Example Task: Producer

```
10
11      //! Task arguments.
12      struct Arguments
13      {
14        //! PRNG type.
15        std::string prng_type;
16        //! PRNG seed.
17        int prng_seed;
18        //! Mean temperature value.
19        float mean_value;
20        //! Standard deviation of temperature measurements.
21        double std_dev;
22      };
23
```

Example Task: Producer

```
24      //! Entry point.
25      struct Task: public Tasks::Periodic
26      {
27        //! PRNG handle.
28        Random::Generator* m_prng;
29        //! Task arguments.
30        Arguments m_args;
```

Example Task: Producer

```
32      //! Task constructor.
33      Task(const std::string& name, Tasks::Context& ctx):
34        Tasks::Periodic(name, ctx),
35        m_prng(NULL)
36      {
37        param("Standard Deviation", m_args.std_dev)
38        .units(Units::Meter)
39        .defaultValue("0.1");
40
41        param("PRNG Type", m_args.prng_type)
42        .defaultValue(Random::Factory::c_default);
43
44        param("PRNG Seed", m_args.prng_seed)
45        .defaultValue("-1");
46
47        param("Mean Value", m_args.mean_value)
48        .defaultValue("25.0")
49        .units(Units::DegreeCelsius)
50        .description("Mean temperature value");
51      }
```

Example Task: Producer

```
53      //! Acquire resources.
54      void
55      onResourceAcquisition(void)
56      {
57        m_prng = Random::Factory::create(m_args.prng_type,
58                                         m_args.prng_seed);
59      }
60
61      //! Release resources.
62      void
63      onResourceRelease(void)
64      {
65        Memory::clear(m_prng);
66      }
```

Example Task: Producer

```
68          //! Periodic work.
69          void
70          task(void)
71          {
72            IMC::Temperature temperature;
73            temperature.value = m_args.mean_value
74                              + m_prng->gaussian()
75                              * m_args.std_dev;
76            dispatch(temperature);
77          }
78        };
79      }
80    }
81
82    DUNE_TASK
83
```

Configuration File

- http://goo.gl/n4nli4
- Configuration file etc/development/workshop.ini:

```
[Require ../common/transports.ini]

[Workshop.Producer]
Enabled      = Always
Entity Label = Producer

[Workshop.Consumer]
Enabled      = Always
Entity Label = Consumer

[Transports.Logging]
Enabled      = Always
Entity Label = Logger
Transports   = Temperature
```

Runtime Output

Command: ./dune -c development/workshop

```
[2014/11/16 19:51:26] - MSG [Daemon] >> system name: 'unknown' (65535)
[2014/11/16 19:51:26] - MSG [Daemon] >> registered tasks: 160
[2014/11/16 19:51:26] - MSG [Daemon] >> base folder: '/home/rasm/tutorial/build'
[2014/11/16 19:51:26] - MSG [Daemon] >> configuration folder: '/home/rasm/tutorial/dune/etc'
[2014/11/16 19:51:26] - MSG [Daemon] >> web server folder: '/home/rasm/tutorial/dune/www'
[2014/11/16 19:51:26] - MSG [Daemon] >> log folder: '/home/rasm/tutorial/build/log/unknown'
[2014/11/16 19:51:26] - MSG [Daemon] >> library folder: '/home/rasm/tutorial/build'
[2014/11/16 19:51:26] - MSG [Daemon] >> firmware folder: '/home/rasm/tutorial/dune/firmware'
[2014/11/16 19:51:26] - MSG [Transports.Cache] >> starting
[2014/11/16 19:51:26] - MSG [Transports.FTP] >> starting
[2014/11/16 19:51:26] - MSG [Transports.Fragments] >> starting
[2014/11/16 19:51:26] - MSG [Transports.HTTP] >> starting
[2014/11/16 19:51:26] - MSG [Transports.LogBook] >> starting
[2014/11/16 19:51:26] - MSG [Transports.Logging] >> starting
[2014/11/16 19:51:26] - MSG [Workshop.Consumer] >> starting
[2014/11/16 19:51:26] - MSG [Workshop.Producer] >> starting
```

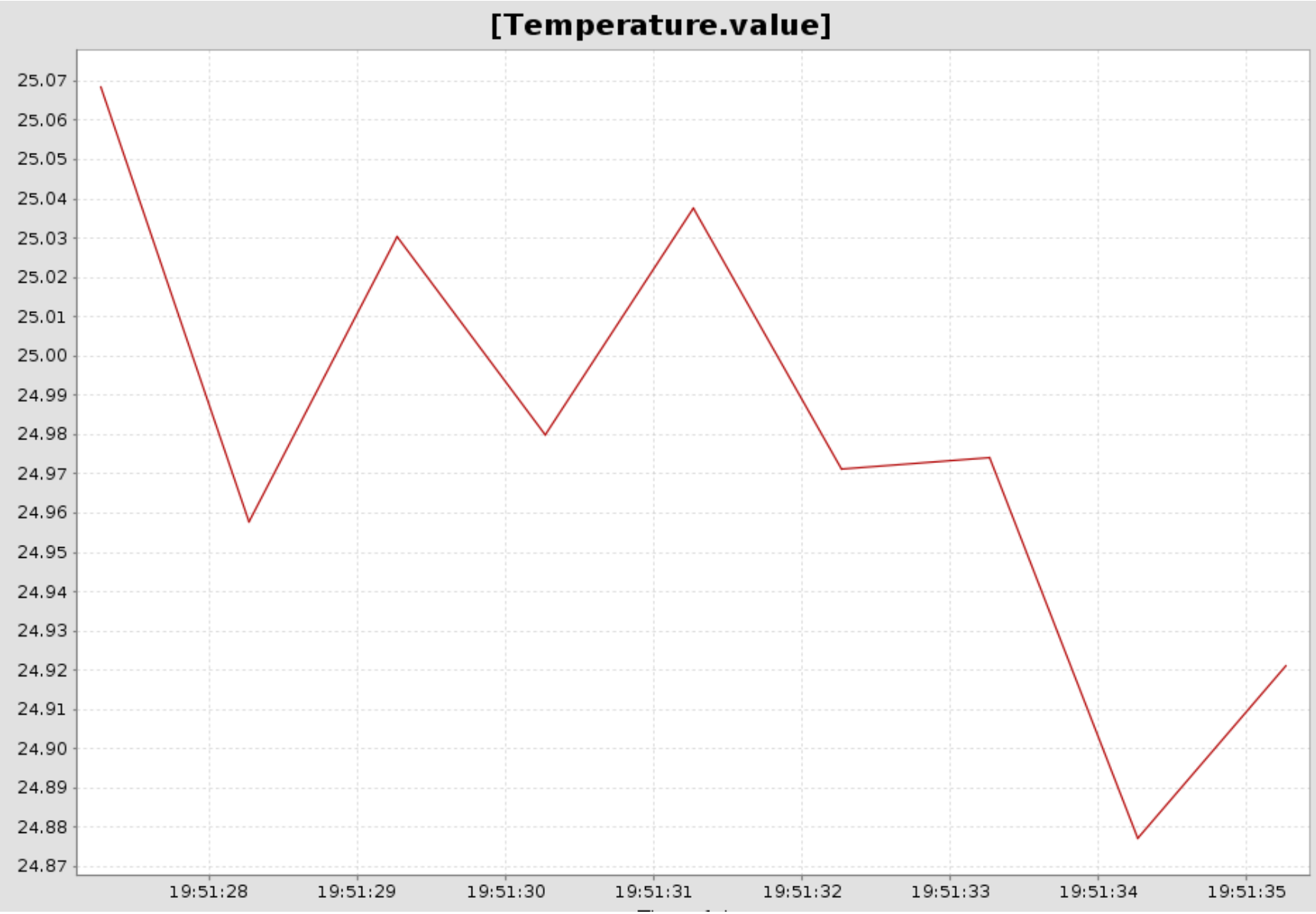
Runtime Output

```
[2014/11/16 19:51:26] - MSG [Transports.HTTP] >> listening on 0.0.0.0:8080
[2014/11/16 19:51:26] - MSG [Transports.Logging] >> log started '20141116/195126'
[2014/11/16 19:51:26] - MSG [Transports.FTP] >> listening on 127.0.0.1:30021
[2014/11/16 19:51:26] - MSG [Transports.FTP] >> listening on 192.168.1.178:30021
[2014/11/16 19:51:26] - MSG [Transports.FTP] >> listening on 10.0.254.1:30021
[2014/11/16 19:51:27] - MSG [Workshop.Consumer] >> temperature is 25.068323
[2014/11/16 19:51:28] - MSG [Workshop.Consumer] >> temperature is 24.957678
[2014/11/16 19:51:29] - MSG [Workshop.Consumer] >> temperature is 25.030371
[2014/11/16 19:51:30] - MSG [Workshop.Consumer] >> temperature is 24.979784
[2014/11/16 19:51:31] - MSG [Workshop.Consumer] >> temperature is 25.037634
[2014/11/16 19:51:32] - MSG [Workshop.Consumer] >> temperature is 24.971085
[2014/11/16 19:51:33] - MSG [Workshop.Consumer] >> temperature is 24.974072
[2014/11/16 19:51:34] - MSG [Workshop.Consumer] >> temperature is 24.877298
```

Log Files

- **DUNE stores log files in the IMC serialization format:**
  - Binary format
  - 1 file for all messages and message types (Data.lsf)
  - Messages are stored roughly in the same order as they were created
  - Supports Gzip and Bzip2 compression (Data.lsf.gz, Data.lsf.bz2)

Log File (Neptus MRA)



Extending Neptus

Requirements for Extending Neptus

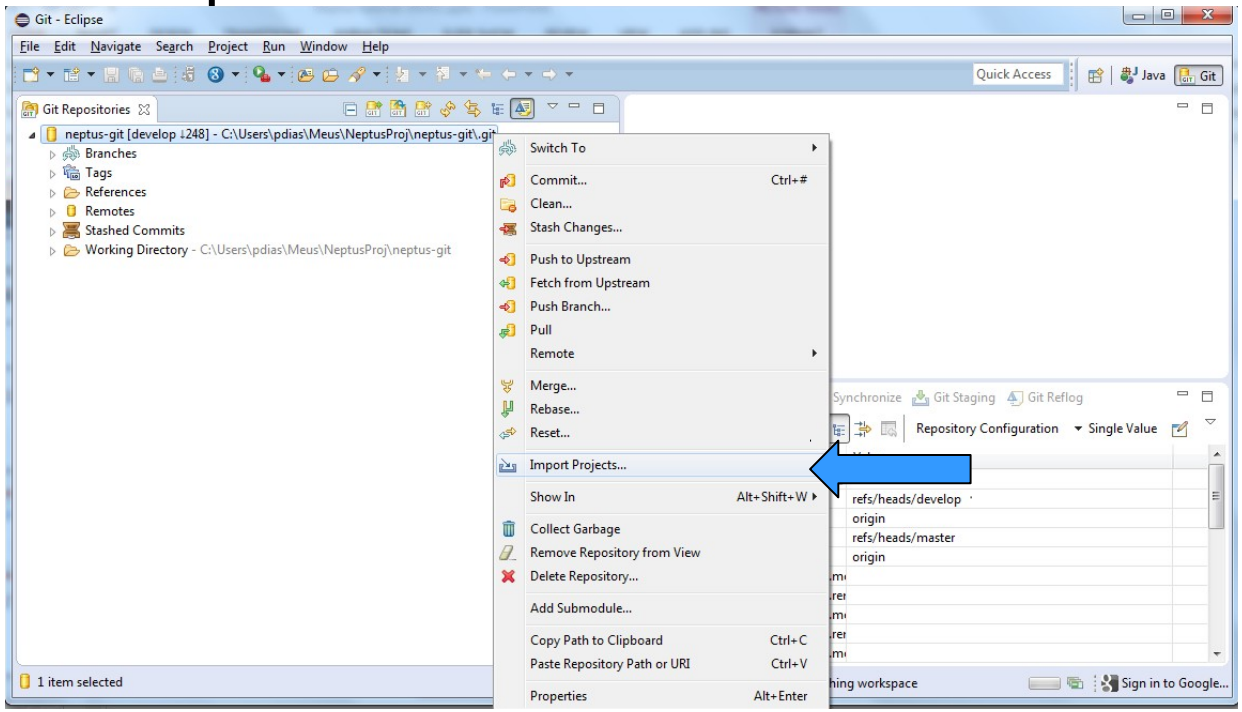
- Installation of Oracle's Java JDK version 8
- Git (Source Control Management)
- Ant (Build System)
- Eclipse Luna for Java Developers

Setup Your Development Tool

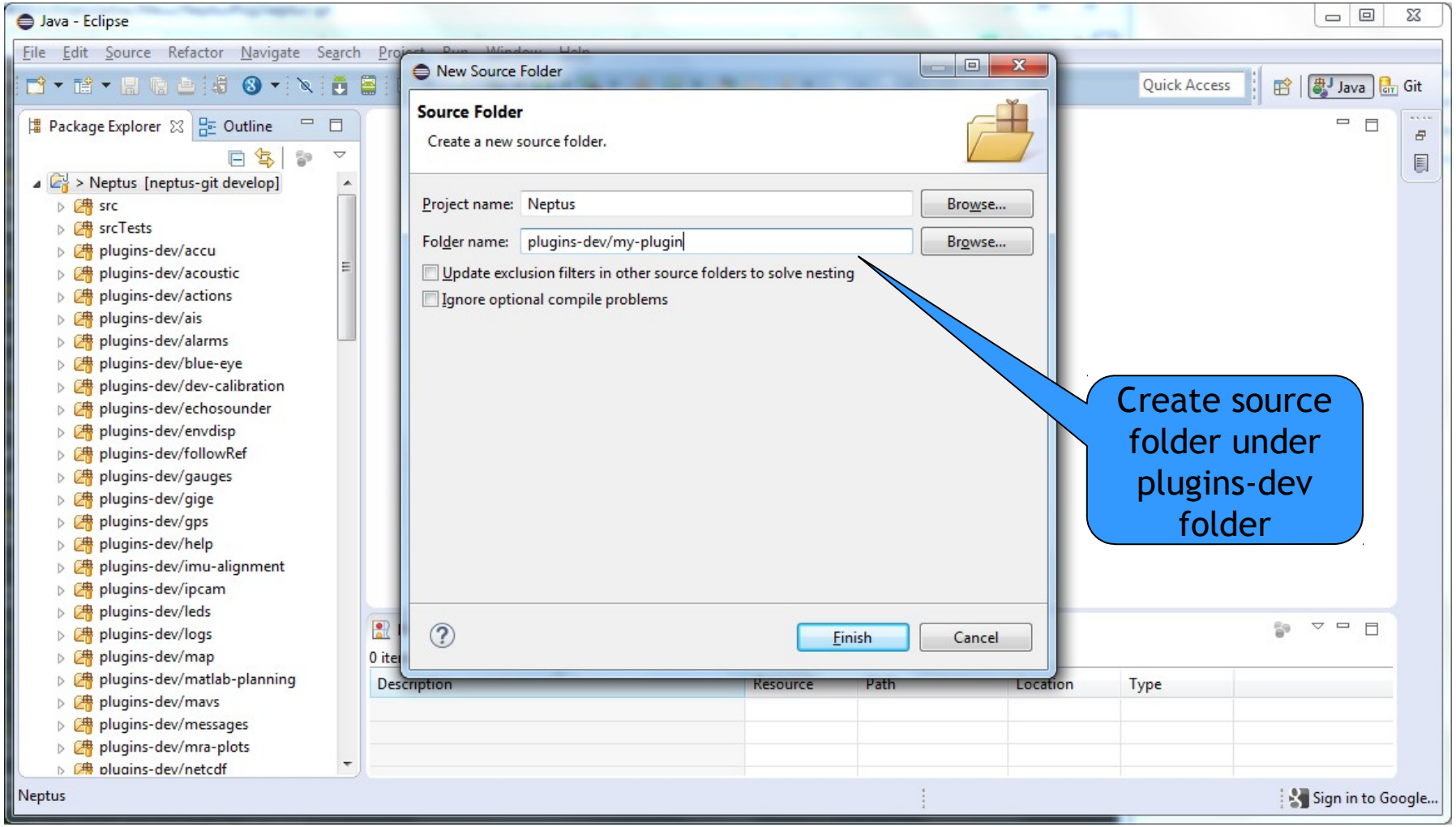
- Clone Neptus
  - Use your favorite Git client to clone Neptus:

    git clone https://github.com/LSTS/neptus.git

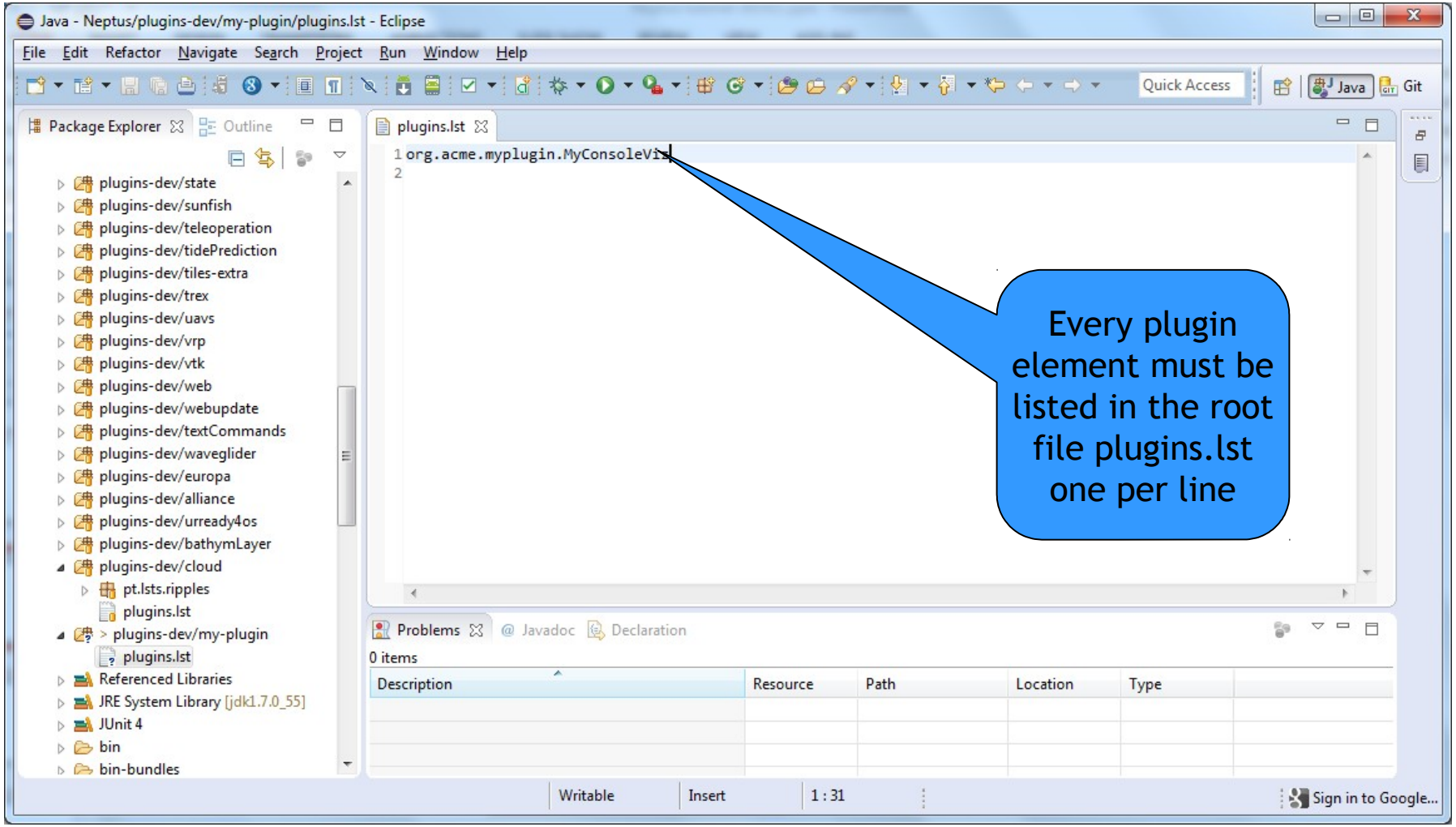
- Configure Eclipse



Creating a plug-in



Plug-in properties



Plug-in example - Console Widget

```
package org.acme.myplugin;
import pt.lsts.neptus.console.ConsoleLayout;
…
/**
 * @author You
 *
 */
@PluginDescription(name = "My Console Viz")
@Popup(pos = POSITION.RIGHT, width = 200, height = 200, accelerator = 'Y')
@SuppressWarnings("serial")
public class MyConsoleViz extends ConsolePanel {

    /**
     * @param console
     */
    public MyConsoleViz(ConsoleLayout console) {
        super(console);
    }

    @Override
    public void initSubPanel() {
    }

    @Override
    public void cleanSubPanel() {
    }
}
```

Every plugin is anotated with PluginDescription

Optionally the panel may be a popup dialog instead of living in the main window

Base console widget extends ConsolePanel

Plug-in example - Console Widget

```
package org.acme.myplugin;
import pt.lsts.neptus.console.ConsoleLayout;
…
/**
 * @author You
 *
 */
@PluginDescription(name = "My Console Viz")
@Popup(pos = POSITION.RIGHT, width = 200, height = 200, accelerator = 'Y')
@SuppressWarnings("serial")
public class MyConsoleViz extends ConsolePanel {

    /**
     * @param console
     */
    public MyConsoleViz(ConsoleLayout console) {
        super(console);
    }

    @Override
    public void initSubPanel() {
    }

    @Override
    public void cleanSubPanel() {
    }
}
```

Every plugin is anotated with PluginDescription

Optionally the panel may be a popup dialog instead of living in the main window

Base console widget extends ConsolePanel

Plug-in example - Console Widget

```
package org.acme.myplugin;
import pt.lsts.neptus.console.ConsoleLayout;
…
/**
 * @author You
 *
 */
@PluginDescription(name = "My Console Viz")
@Popup(pos = POSITION.RIGHT, width = 200, height = 200, accelerator = 'Y')
@SuppressWarnings("serial")
public class MyConsoleViz extends ConsolePanel {

    /**
     * @param console
     */
    public MyConsoleViz(ConsoleLayout console) {
        super(console);
    }

    @Override
    public void initSubPanel() {
    }

    @Override
    public void cleanSubPanel() {
    }
}
```

Every plugin is anotated with PluginDescription

Optionally the panel may be a popup dialog instead of living in the main window

Base console widget extends ConsolePanel

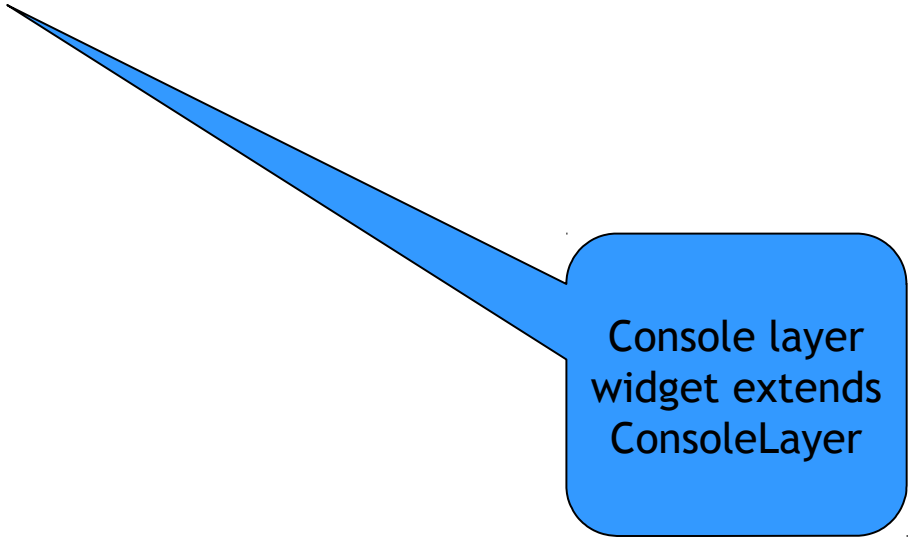
Plugin example - Map Layer

```
package org.acme.myplugin;
import pt.lsts.neptus.console.ConsoleLayer;
…
/**
 * @author You
 *
 */
@PluginDescription(name = "My Console Layer")
@LayerPriority(priority = 66)
public class MyConsoleLayer extends ConsoleLayer {
    public MyConsoleLayer() {
    }

    @Override
    public void initLayer() {
    }

    @Override
    public void cleanLayer() {
    }

    @Override
    public boolean userControlsOpacity() {
        return false;
```



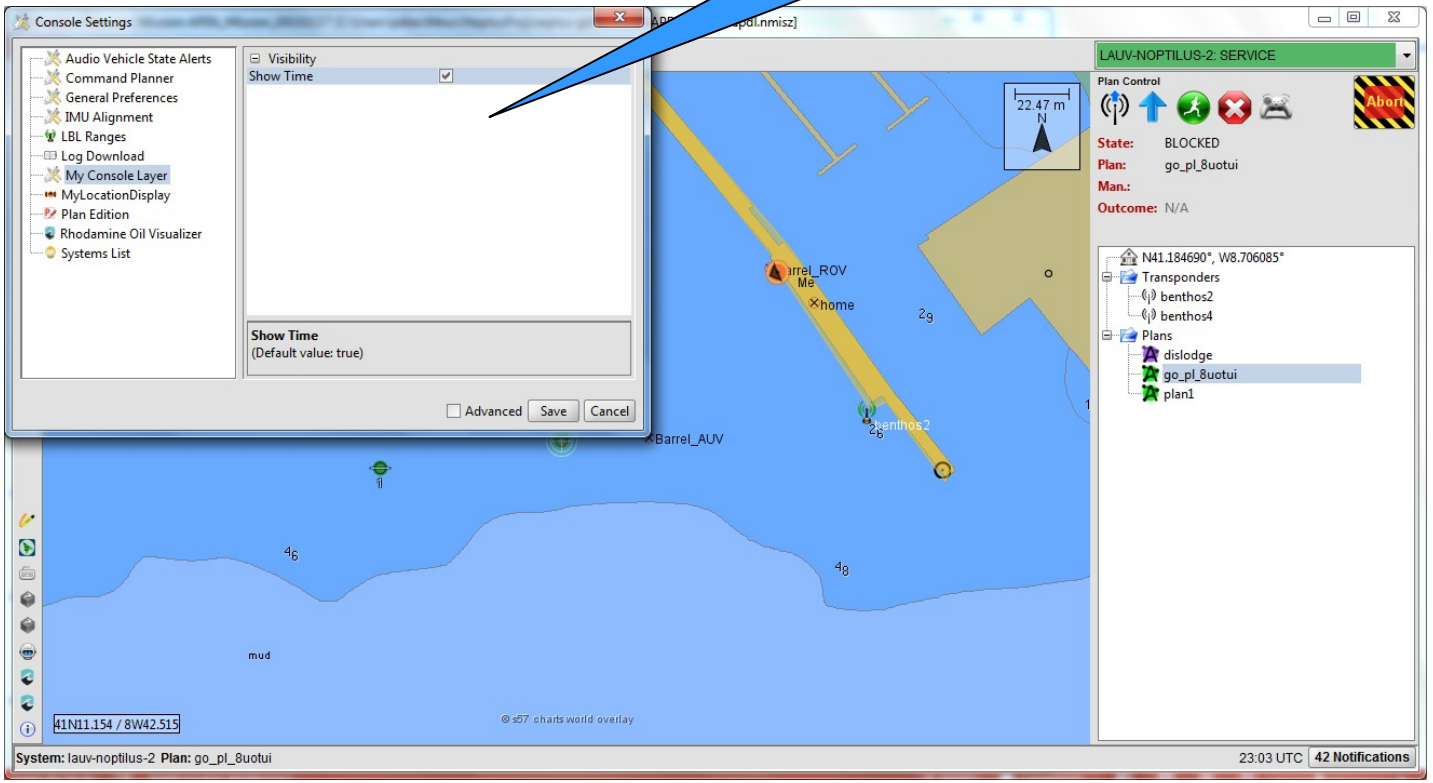
Plug-in example - Map Layer

```
…
public class MyConsoleLayer extends ConsoleLayer {

    @NeptusProperty(name = "Show Time", userLevel = LEVEL.REGULAR,
            category="Visibility", editable = true)
    public boolean showTime = true;

    public MyConsoleLayer() {
    }
…
```

Adding properties for the operator to change



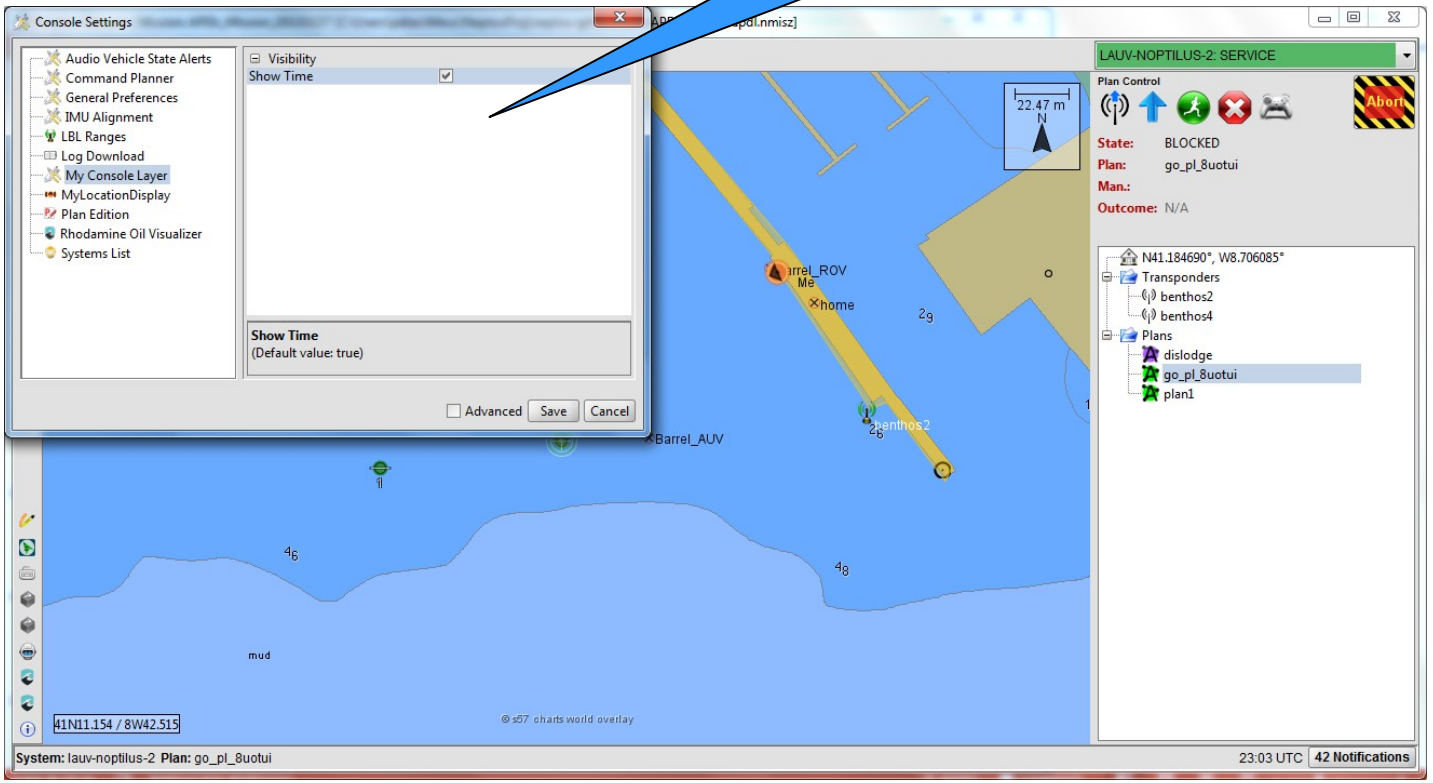
Plug-in example - Map Layer

```
…
public class MyConsoleLayer extends ConsoleLayer {

    @NeptusProperty(name = "Show Time", userLevel = LEVEL.REGULAR,
            category="Visibility", editable = true)
    public boolean showTime = true;

    public MyConsoleLayer() {
    }
…
```

Adding properties for the operator to change



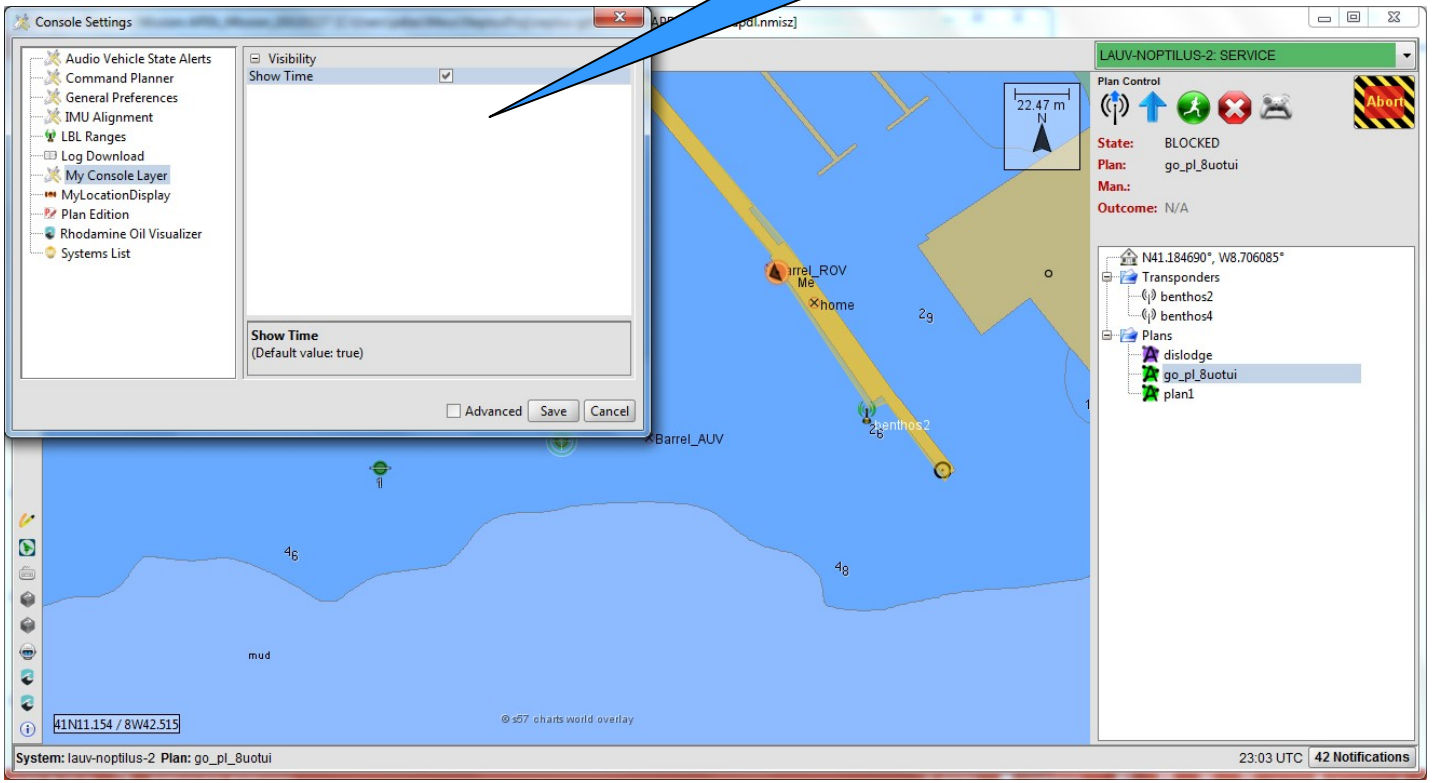
Plug-in example - Map Layer

```
…
public class MyConsoleLayer extends ConsoleLayer {

    @NeptusProperty(name = "Show Time", userLevel = LEVEL.REGULAR,
            category="Visibility", editable = true)
    public boolean showTime = true;

    public MyConsoleLayer() {
    }
…
```

Adding properties for the operator to change



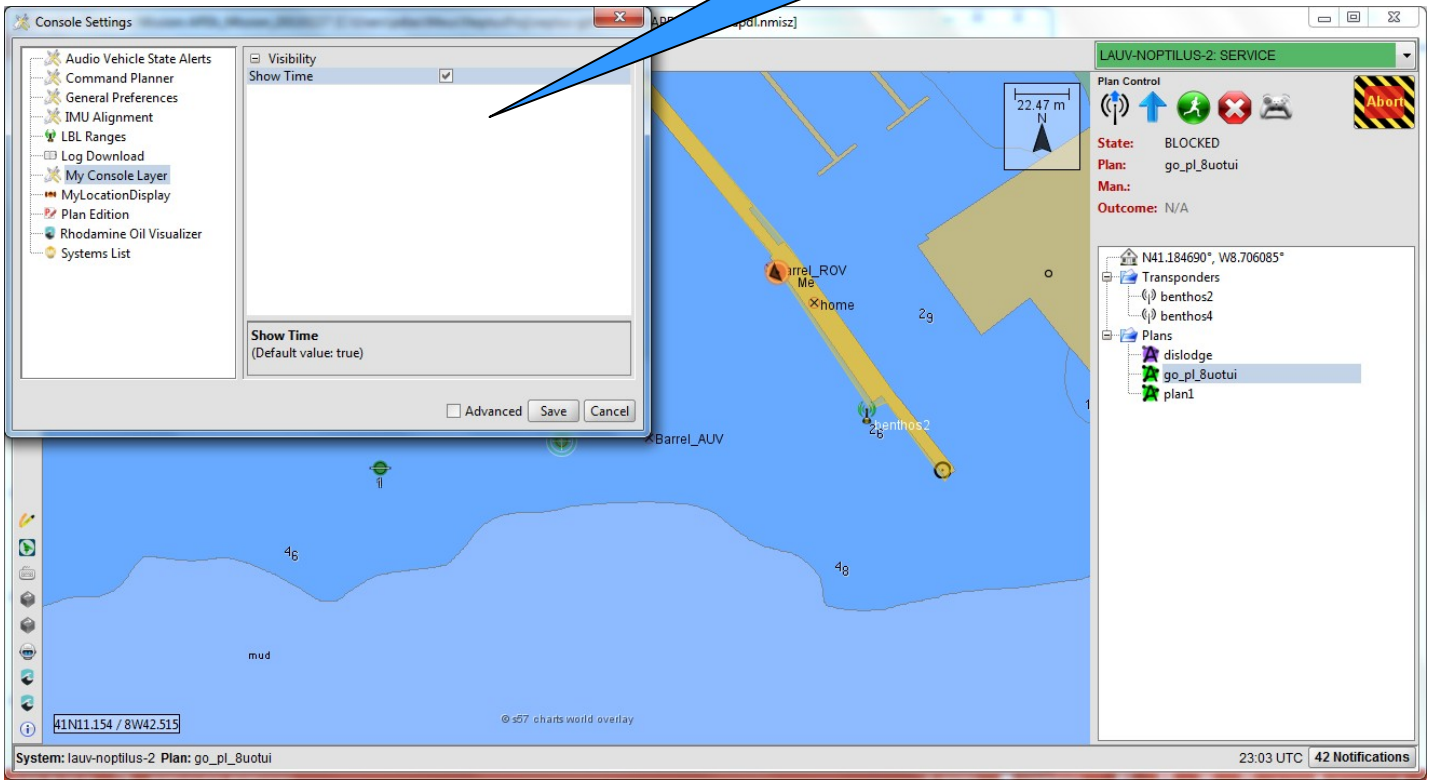
Plug-in example - Map Layer

```
…
public class MyConsoleLayer extends ConsoleLayer {

    @NeptusProperty(name = "Show Time", userLevel = LEVEL.REGULAR,
            category="Visibility", editable = true)
    public boolean showTime = true;

    public MyConsoleLayer() {
    }
…
```

Adding properties for the operator to change



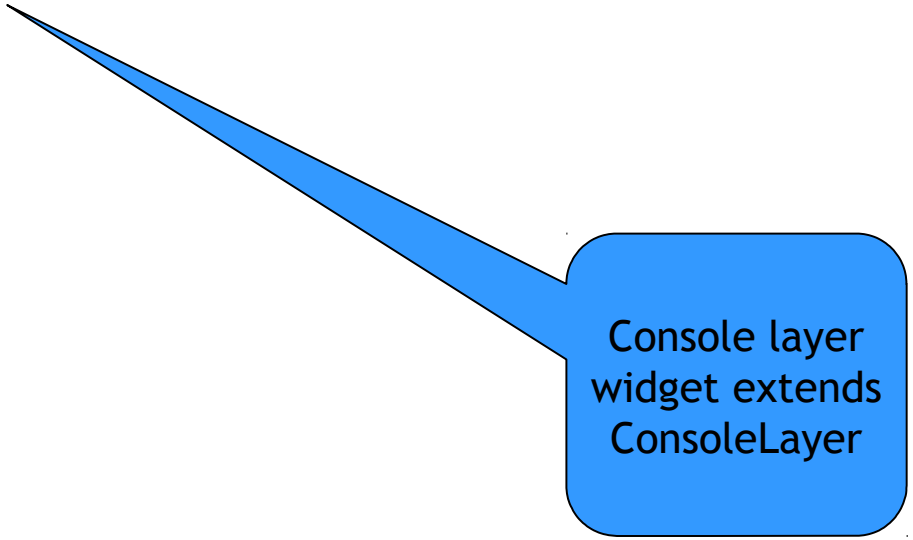
Plugin example - Map Layer

```
package org.acme.myplugin;
import pt.lsts.neptus.console.ConsoleLayer;
…
/**
 * @author You
 *
 */
@PluginDescription(name = "My Console Layer")
@LayerPriority(priority = 66)
public class MyConsoleLayer extends ConsoleLayer {
    public MyConsoleLayer() {
    }

    @Override
    public void initLayer() {
    }

    @Override
    public void cleanLayer() {
    }

    @Override
    public boolean userControlsOpacity() {
        return false;
```



Packaging the plug-in

- By using Ant you can compile all
  - ant
- It will create a jar in plugins folder name my-plugin.jar
- To add to console to test
  - Run pt.lsts.neptus.loader.NeptusMain
  - Open lauv.ncon console
  - Click menu View>Plugin Manager add your plugins elements and save console