|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| ESS – JELS – Users guide |
|  |
|  |

JELS - Technical specification

Document revision history

|  |  |  |
| --- | --- | --- |
| Version | Reason for revision | Date |
| 1.0 | New Document | 7 January 2014 |

Table of Contents

1. References – relavant documentation 4

2. Introduction 4

2.1 Purpose 4

2.2 Definitions, Acronyms and Abbreviations 4

3. General Architecture 5

4. Setting up the Environment 5

4.1 Eclipse 5

4.2 RPMs 5

4.3 Source code 5

5. Importing lattice 5

6. Running 5

6.1 Setting up initial parameters 5

6.2 Setting up different algorithms 5

7. Output 5

8. Additional information 5

# References

The relevant documentation is

1. EPICS (<http://www.aps.anl.gov/epics>)
2. XALConfigurationAndIntialization-2ndEd.pdf (<http://sourceforge.net/projects/xaldev/files/>)
3. OpenXAL reference manual

(<https://hg.esss.dk/ad/physics-core/high-level-apps/OpenXAL/documentation/reference-manual/raw-file/tip/DOC-ESS-OpenXAL-ReferenceManual.pdf>)

1. ESS wiki pages on OpenXAL (<https://twiki.esss.dk/ad/index.php/OpenXAL>)

Links to source code:

1. JELS (<https://git.esss.dk/ad/pc-hla-jels>)
2. OpenXAL – ESS copy (<https://git.esss.dk/ad/openXAL>)
3. OpenXAL – original source (<http://sourceforge.net/p/xaldev/openxal/ci/master/tree/>)

# Introduction

JELS stands for Java ESS Linac simulator and is an extension of OpenXAL with basic elements and physics, which was imported from ELS, and is needed to run the simulation.

## Purpose

The purpose of this document is to introduce JELS to a user which has physics background, teach him to setup and run simulation and gather the outputs. Additionally there are instructions how to simulate an accelerator, how to tweak some of the parameters and run an online simulation.

## Definitions, Acronyms and Abbreviations

| Term | Abbreviation | Definition |
| --- | --- | --- |
| ELS | ESS Linac simulator |  |
| JELS | Java ELS |  |
| SMF | Standard Machine Framework |  |

# General Architecture

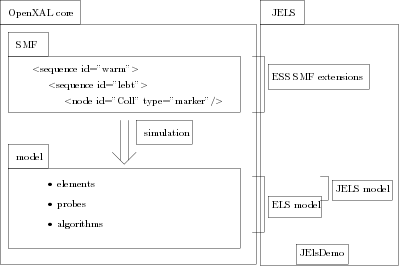


Figure 1: General architecture of OpenXAL core and JELS

JELS depends only on the core part of the OpenXAL.

Core part of OpenXAL is composed of SMF, model and simulation.

SMF, i.e. standard machine framework is a hierarchical object model, representing the high level logical structure of the accelerator.

Hierarchy in SMF is represented with sequences (implemented by java class AcceleratorSeq) and are arbitrarily nested. Examples of sequences are accelerator sections such as LEBT, MEBT, etc. as well as higher structure (warm, cold) and more detail one (i.e. LEBT is broken into lebt-slot, rfq).

At the bottom of the hierarchy in SMF are so called nodes (implemented by java class AcceleratorNode and all of its extensions). Nodes are of different types: quadrupole, dipole, RF gap. Nodes contain the parameters describing the devices they represent. The nodes do not contain any code needed doing actual computations needed in the simulation.

Model part of the OpenXAL contains everything needed to run a simulation. Provided are interfaces for elements, probes and algorithms (and also some implementations).

Elements are paired with SMF nodes, adding additional physics description to them and needed numerical calculations (optics transfer matrices). They are contained in linear list and have no hierarchy and are stateless.

Probes contain description of physical state of the particle or bunch, depending on the algorithm that is used.

Algorithms take care of propagating the probe through list of elements and updating the probe’s state. At each point of the simulation, the state of the probe is saved, thus forming a trajectory.

Simulation (i.e. xal.sim java package) provides the necessary “glue” between the SMF and model part.

For more information about OpenXAL please refer to [3] OpenXAL reference manual.

JELS extends OpenXAL’s SMF bending magnets and RF cavities with additional parameters. The extensions are in the Java package eu.ess.jels.smf.

JELS supports two additional models to the default OpenXAL’s:

* ELS model – basically the same model as in ELS ported from C to Java including its own probe and algorithm
* JELS model – ELS model adjusted to work with probes and algorithms provided by OpenXAL

For ELS model the whole triple is implemented: custom elements, probe and algorithm and they’re located in packages eu.ess.jels.model.elem.els, eu.ess.jels.model.probe and eu.ess.jels.model.alg respectively.

For JELS only modelling elements are implemented in package eu.ess.jels.model.elem.jels.

At last a simple application to run the simulation is available in eu.ess.jels.JElsDemo class.

We conclude with additional list of expressions used throughout the code:

| Classes | Description |
| --- | --- |
| Probe | base class containing physical state of the particle or bunch, depending on the algorithm that is used |
| ProbeState | basically same thing as a probe (to add some mess every specific probe has a sibling ProbeState) |
| Trajectory | basically List<ProbeState>, with some more functionality (here mess is added by extending Trajectory for each probe type, but not adding anything) |
| Lattice | basically List<Element>, with methods that help algorithm propagate the probe |
| LineModel, ElementSeq | same thing as Lattice |
| ElementMapping | class describes mapping of SMF nodes to model elements |

Table 1: List of relevant base classes in OpenXal

# Setting up the Environment

## Linux – RPMs

* Link to the repository
* How to configure the repository
* Shell instructions to install needed repos

## Other operating system

* Install Java
* Install Eclipse
* Install Git
* Download Git
* Checkout the code

<https://twiki.esss.dk/ad/index.php/OpenXAL:Setting_up_development_environment>

TODO update Wiki page

TODO add RPM, GIT instructions to Wiki

## Importing source code to Eclipse

Prerequisites for this step are:

* Eclipse is installed
* JELS source is downloaded to a local folder

Do the following:

1. Start Eclipse
2. File > Import…
3. Maven > Existing Maven Projects
4. Choose the folder where JELS is located
5. Finish – you should be able to see “jels” project on the left side of the screen in Project Explorer. You can already choose eu.ess.jels.JElsDemo file and run it.

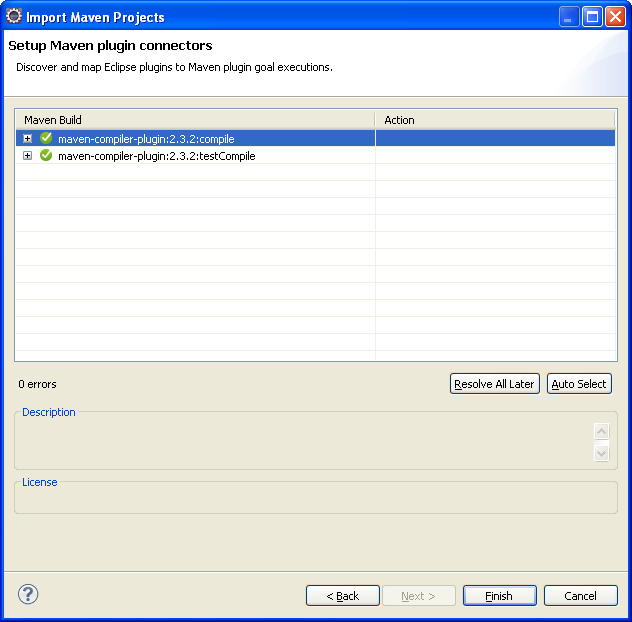
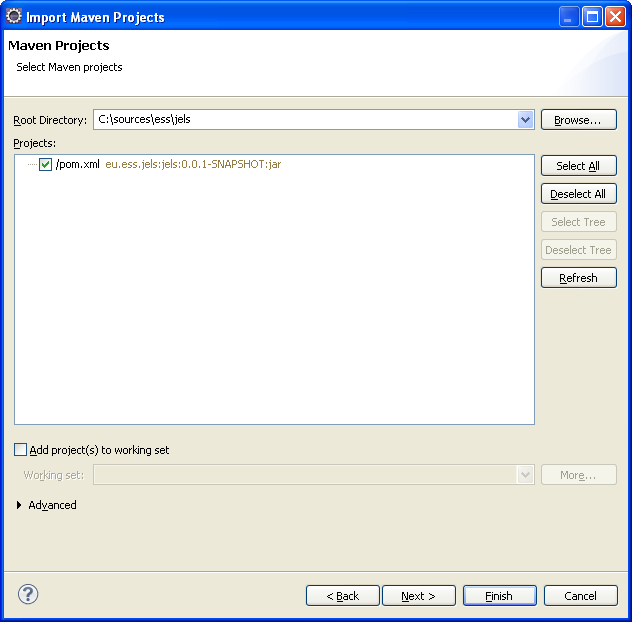
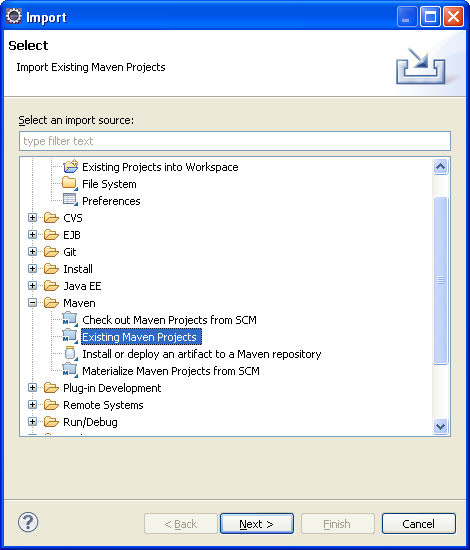


Figure 2: Eclipse wizard guiding the import of JELS maven project

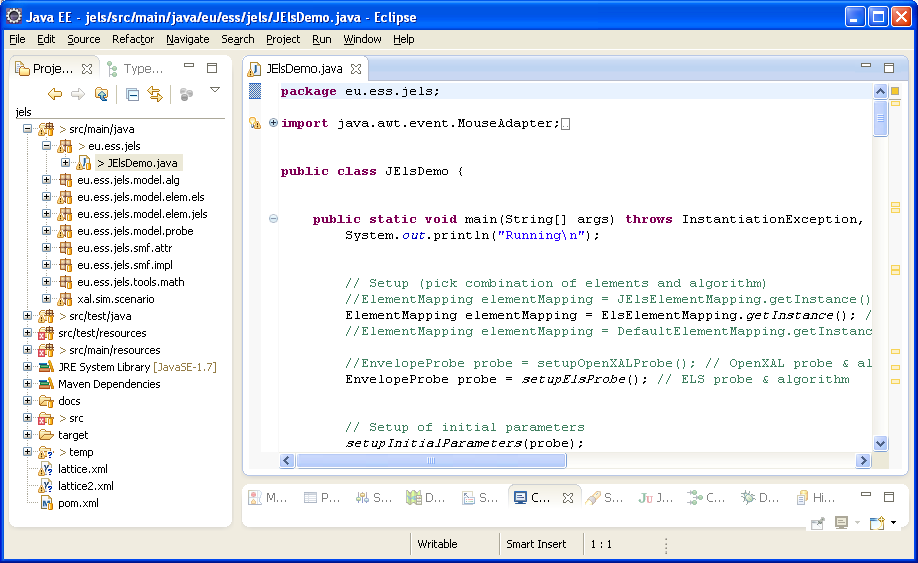


Figure 3: Imported JELS project in Eclipse

# Input files

OpenXAL needs minimally following three files to run:

* main.xal – main configuration referencing to other two files
* ess.impl – configures mapping between xml type attribute and smf device implementation
* ess.xdxf – **optics file, defining the hardware configuration of the accelerator**

All three files are already provided in JELS in folder src/main/resource.

File ess.xdxf is the main point of interest. It contains the description of the accelerator in a readable XML format. For more information about the file format and parameters please refer to [3] OpenXAL reference manual.

TODO generating ess.xdxf with LatticeGenerator

# Running

To execute the simulation please refer to JElsDemo class, where all the necessary steps are made. Following sections only explain the key steps that are used.

## Setting up different algorithms

Only RMS envelope propagation was tested. To try out other algorithm please refer to [3] OpenXAL reference manual, section 5.3.

Algorithm together with a corresponding probe is setup in the following lines of code located in JElsDemo.java file:

**// Setup (pick combination of elements and algorithm)**

//ElementMapping elementMapping = JElsElementMapping.getInstance(); // JELS element mapping - transfer matrices in OpenXal reference frame

ElementMapping elementMapping = ElsElementMapping.*getInstance*(); // ELS element mapping - transfer matrices in TraceWin reference frame

//ElementMapping elementMapping = DefaultElementMapping.getInstance(); // OpenXAL element mapping - transfer matrices in OpenXal reference frame

//EnvelopeProbe probe = setupOpenXALProbe(); // OpenXAL probe & algorithm

EnvelopeProbe probe = *setupElsProbe*(); // ELS probe & algorithm

Example 1: Setup of algorithm used in JElsDemo class

Several combinations of element mapping and algorithms are possible. Here is a brief description of the outcome:

| Element mapping | Algorithm | Description |
| --- | --- | --- |
| Default | setupOpenXALProbe() | OpenXAL’s transfer matrices and dynamics. Most of the matrices are flawed at ESS’s energies, besides there are bugs in RF gaps implementation, producing incorrect results for NCELLS element. |
| Els | setupElsProbe() | ELS’s transfer matrices and ELS dynamics. The ELS propagates only central matrix elements – the visible difference happens at dipole magnets. |
| Els | setupOpenXALProbe() | ELS’s transfer matrices (in laboratory frame) and OpenXAL’s dynamics. |
| JEls | setupOpenXALProbe() | JELS’s transfer matrices(in beam reference frame) and OpenXAL’s dynamics |

Table 1: Combining element mapping and algorithm and respective results.

## Setting up initial conditions

As it should be clear from section on General Architecture, the initial conditions are set on the probe. This can be done programmatically in Java or using a probe file. Probes from different algorithms are compatible, so that the initial conditions can be set at a single spot:

**public** **static** **void** setupInitialParameters(EnvelopeProbe probe)

{

probe.setSpeciesCharge(-1);

probe.setSpeciesRestEnergy(9.3829431e8);

//elsProbe.setSpeciesRestEnergy(9.38272013e8);

probe.setKineticEnergy(3e6);//energy

probe.setPosition(0.0);

probe.setTime(0.0);

**double** beta\_gamma = probe.getBeta() \* probe.getGamma();

probe.initFromTwiss(**new** Twiss[]{

**new** Twiss(-0.1763,0.2442,0.2098\*1e-6 / beta\_gamma),

**new** Twiss(-0.3247,0.3974,0.2091\*1e-6 / beta\_gamma),

**new** Twiss(-0.5283,0.8684,0.2851\*1e-6 / beta\_gamma)});

probe.setBeamCurrent(0.0);

//probe.setBeamCurrent(50e-3);

// probe.setBunchFrequency(4.025e8);

}

Example 2: Setting up initial conditions

Another option is setting up initial conditions using a file. The example of a initial state file mebt-initial-state.xml may be found in folder src/main/resource. The example may be used by changing the call to setupInitialParameters() with loadInitialParameters() in JElsDemo.java file. For additional reference on file format please refer to [3] OpenXAL reference manual, section 5.5.1.

<?xml version = *'1.0'* encoding = *'UTF-8'*?>

<state id=*""* type=*"xal.model.probe.traj.EnvelopeProbeState"*>

<species Er=*"9.3829431E8"* q=*"-1.0"*/>

<location W=*"3000000.0"* elem=*""* s=*"0.0"* t=*"0.0"*/>

<beam I=*"0.0"* f=*"0.0"* phase=*"(0.0,0.0,0.0)"*/>

<envelope

alphaX=*"-0.1763"* betaX=*"0.2442"* emitX=*"2.6215161390E-6"*

alphaY=*"-0.3247"* betaY=*"0.3974"* emitY=*"2.6127694216E-6"*

alphaZ=*"-0.5283"* betaZ=*"0.8684"* emitZ=*"3.5624130182E-6"*/>

</state>

Example 3: Initial parameters specified in a file

# Modifying parameters of the elements in Optics file

Although it might be inviting to modify some parameters directly in the optics file, this might be limiting in some applications, especially when such modifications need to be automatized.

Alternative method is doing it programmatically using so called “synchronization manager”. This method is limited to dynamic parameters, such as magnetic fields and amplitudes of RF cavities. Other parameters still need to be modified directly in the file or on the SMF object model.

Here a simple code snippet is provided, but for more information please refer to [3] OpenXAL reference manual, section 5.6.

// Manually changing magnet field

AcceleratorNode mag = sequence.getNodeWithId("MEBT-PBO\_QV-1");

scenario.setModelInput(mag,

ElectromagnetPropertyAccessor.*PROPERTY\_FIELD*, 15);

scenario.resync();

// Manually changing rfcavity parameters

AcceleratorNode rfcavity = sequence.getNodeWithId("DTL-TANK-1-CELL-1");

scenario.setModelInput(rfcavity,

RfCavityPropertyAccessor.*PROPERTY\_AMPLITUDE*, 17145); //was 17145.76

scenario.setModelInput(rfcavity,

RfCavityPropertyAccessor.*PROPERTY\_PHASE*, -34); // was -35

scenario.resync(); // all rfgaps are updated

Example 4: Manually setting properties of lattice elements via SynchronizationManager

# Output

The results of the simulation are contained in a trajectory which is basically a list of probe states. A probe state has all the needed methods to get the basic values as well the values that are based on those.

One of the basic tasks is to traverse the whole trajectory and output the results. How it is done can be seen in the following code snippet:

// Getting results

Trajectory trajectory = probe.getTrajectory();

Iterator<ProbeState> iterState = trajectory.stateIterator();

**while** (iterState.hasNext())

{

EnvelopeProbeState ps =

(EnvelopeProbeState) iterState.next();

Twiss[] twiss = ps.twissParameters();

System.*out*.printf("%E %E %E %E %E %E %E %E %E %E %E\n",

ps.getPosition(), ps.getGamma()-1,

twiss[0].getEnvelopeRadius(),twiss[0].getBeta(),

twiss[1].getEnvelopeRadius(),twiss[1].getBeta(),

twiss[2].getEnvelopeRadius(),twiss[2].getBeta(),

twiss[2].getBeta()/Math.*pow*(ps.getGamma(), 2),

ps.getTime(), ps.getKineticEnergy());

}

Example 5: Printing results to standard output

Another common task is retrieving values at a certain point. This can be done using methods stateForElement and stateAtPosition of the trajectory:

ProbeState ps1 = trajectory.stateForElement("MEBT-PBI\_BPM-2a");

ProbeState ps2 = trajectory.stateAtPosition(101.2);

Example 6: Retrieving probe states at certain element or at certain position.

# Compiling and running from Shell

TODO