

# OOPIC Pro

OOPIC Pro <sup>TM</sup>

User's Guide

Version 2.0.2

Tech-X Corporation  
5621 Arapahoe Avenue, Suite A  
Boulder, CO 80303  
<http://www.txcorp.com>  
[sales@txcorp.com](mailto:sales@txcorp.com)



<i>CONTENTS</i>	2
-----------------	---

## **Contents**

<b>1 Preface</b>	<b>7</b>
1.1 Conventions Used . . . . .	7
1.2 Copyright Information . . . . .	7
1.2.1 OOPIC Kernel Copyright Information . . . . .	7
1.2.2 HDF5 Copyright Information . . . . .	8
1.2.3 JPEG Copyright Information . . . . .	9
1.2.4 PETSc Copyright information . . . . .	9
<b>2 Introduction</b>	<b>11</b>
2.1 Objective of This Guide . . . . .	11
2.2 Organization of This Guide . . . . .	11
<b>3 Getting Started</b>	<b>13</b>
3.1 Installation . . . . .	13
3.2 Evaluation Version of OOPIC Pro . . . . .	13
<b>4 The Graphical User Interface (GUI)</b>	<b>14</b>
4.1 Launching the GUI . . . . .	14
4.2 Providing Input Data . . . . .	14
4.3 Using the GUI to Control the Simulation . . . . .	14
4.3.1 The Control Window . . . . .	14
4.3.2 The Control Window Menu Bar . . . . .	15
4.3.3 The Control Window Control Buttons . . . . .	16
4.4 Displaying Diagnostic Plots . . . . .	17
4.5 Plot Types . . . . .	17
4.5.1 2-D Particle Plots (configuration space) . . . . .	17
4.5.2 2-D Phase Space Plots . . . . .	18
4.5.3 2-D Vector Plots . . . . .	19

<i>CONTENTS</i>	3
4.5.4 3-D Surface Plots . . . . .	19
4.5.5 2-D Line Plots . . . . .	20
4.6 Manipulating Diagnostic Plots . . . . .	20
4.6.1 2-D Plots . . . . .	20
4.6.2 3-D plots . . . . .	22
4.7 Dump Files . . . . .	23
4.7.1 Saving a Dump File . . . . .	23
4.7.2 Saving a Dump File Periodically . . . . .	23
4.7.3 Loading a dump file . . . . .	24
4.8 Movies . . . . .	24
4.9 Common problems . . . . .	24
<b>5 Overview of Input File Structure and Parameter Groups</b>	<b>25</b>
5.1 Description Block . . . . .	25
5.2 Variables Block . . . . .	25
5.3 Region Block . . . . .	26
5.4 Example Input File . . . . .	27
<b>6 Blocks and Parameters in OOPIC Pro Input Files</b>	<b>30</b>
6.1 Syntax, Operators, and Built-in Functions . . . . .	30
6.1.1 Data types . . . . .	30
6.1.2 Floating Point Notation . . . . .	30
6.1.3 Operators . . . . .	31
6.1.4 Constants . . . . .	31
6.1.5 Functions . . . . .	31
6.1.6 xtFlag — Default Settings . . . . .	32
6.2 Overall Simulation Parameters . . . . .	32
6.2.1 Grid . . . . .	32
6.2.2 Control . . . . .	33

<i>CONTENTS</i>	4
6.3 Species blocks and Particle Creation . . . . .	38
6.3.1 Species . . . . .	38
6.3.2 MCC (Monte Carlo Collisions) . . . . .	39
6.3.3 MCC (Monte Carlo Collisions) with User-Defined Cross Section Data . . . . .	41
6.3.4 Load . . . . .	44
6.3.5 VarWeightLoad . . . . .	46
6.3.6 PlasmaSource . . . . .	46
6.4 Boundary Conditions . . . . .	47
6.4.1 Generic Boundary Conditions (Boundary Geometry) . . . . .	48
6.4.2 Generic Boundary Conditions (Boundary Material Properties) . . . . .	50
6.4.3 Dielectric . . . . .	51
6.4.4 Conductor . . . . .	51
6.5 Temperature rise at the boundaries . . . . .	52
6.5.1 Equipotential . . . . .	53
6.5.2 Polarizer . . . . .	54
6.5.3 Secondary . . . . .	55
6.5.4 Secondary2 . . . . .	55
6.5.5 Secondary3 . . . . .	56
6.5.6 Sputter . . . . .	57
6.5.7 DielectricRegion . . . . .	58
6.5.8 DielectricTriangle . . . . .	59
6.5.9 CurrentRegion . . . . .	59
6.5.10 Iloop . . . . .	60
6.5.11 CylindricalAxis . . . . .	60
6.6 Ports . . . . .	60
6.6.1 ExitPort . . . . .	61
6.6.2 Gap . . . . .	61
6.6.3 PortGauss . . . . .	62

<i>CONTENTS</i>	5
6.7 Particle Emitters . . . . .	63
6.7.1 Generic Emitter Parameters . . . . .	63
6.7.2 BeamEmitter . . . . .	64
6.7.3 EmitPort . . . . .	65
6.7.4 VarWeightBeamEmitter . . . . .	65
6.7.5 FowlerNordheimEmitter . . . . .	66
6.8 Diagnostics . . . . .	67
6.8.1 Default Diagnostics . . . . .	67
6.8.2 Temperature rise at the boundaries . . . . .	69
6.8.3 User-Defined Diagnostics . . . . .	70
6.8.4 H5Diagnostic . . . . .	73
6.8.5 Limit . . . . .	74
6.8.6 Relation . . . . .	75
6.8.7 Algebra . . . . .	75
6.8.8 Storing Time History Diagnostics . . . . .	75
<b>7 Example Solutions</b>	<b>77</b>
7.1 A Simple Klystron . . . . .	77
7.2 Field-induced Tunneling Ionization by a Laser Pulse . . . . .	79
7.3 Plasma Wakefield Accelerator . . . . .	88
7.4 Secondary particle production . . . . .	91
<b>8 References</b>	<b>92</b>
<b>9 Advanced Use of OOPIC Pro</b>	<b>93</b>
9.1 Command-Line Options . . . . .	93
9.2 Parallel OOPIC Pro (Linux only) . . . . .	95

<i>CONTENTS</i>	6
<b>10 FAQ — Frequently Asked Questions</b>	<b>98</b>
10.1 Can sputtering (removal of surface atoms due to incident ions) be modeled? . . . . .	98
10.2 Are instabilities in the model more likely related to overly large time step or high np2c? . . . . .	98
10.3 How does OOPIC Pro deal with 3D parameters such as density? In a 2-d simulation, how are these 3D values calculated? . . . . .	98
10.4 Do the emitters and beamemitters reabsorb electrons or the particles they are emitting (i.e. electrons come back to emitter as a result of manipulating the fields)? . . . . .	99
10.5 What are the parameters I need to use to do relativistic simulations in OOPIC Pro? . . . . .	99
<b>A Sample input files</b>	<b>100</b>
A.1 beamplasmatest.inp . . . . .	100
A.2 Two Stream . . . . .	104

# 1 Preface

OOPIC Pro is built on the XOOPIC physics kernel, an X11-based object-oriented particle-in-cell (PIC) code. This simulation was originally developed at the University of California at Berkeley, beginning in 1995, by members of the Plasma Theory and Simulation Group (PTSG). Since 1998, Tech-X Corporation has been working in collaboration with PTSG staff to improve and generalize the XOOPIC physics kernel. Additionally, a cross-platform GUI has been added.

## 1.1 Conventions Used

The symbol “→” will be used to denote selection from a sub-menu, for example:

View → Log → Standard Output

If an ellipsis (“...”) appears at the end of a menu choice, this means that subsequent interaction will be necessary — usually interaction with a dialog box.

Additionally, references to OOPIC Pro input file data, file and path names, and system commands will be printed with fixed-width fonts, such as

```
Species
```

Physical units in this document use the meters/kilograms/seconds (MKS) standard unless otherwise specified.

## 1.2 Copyright Information

OOPIC Pro ©Copyright 1998-2008 by Tech-X Corporation.

All rights reserved. See [http://www.txcorp.com/products/OOPIC\\_Pro/](http://www.txcorp.com/products/OOPIC_Pro/) for licensing information.

OOPIC Pro is based in part on the work of the Plasma Theory and Simulation Group of the University of California at Berkeley, the Independent JPEG Group, and the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC). Their respective copyright notices are reproduced below.

### 1.2.1 OOPIC Kernel Copyright Information

Certain materials incorporated herein are Copyright ©1993-2000, The Regents of the University of California (REGENTS). All Rights Reserved, except for nonexclusive license to Tech-X Corporation for development and distribution of commercial versions.

IN NO EVEN SHALL REGENTS BE LIABLE TO ANY PART FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USER OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF REGENTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

REGENTS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED “AS IS”. REGENTS HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

## 1.2.2 HDF5 Copyright Information

HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 2006-2008 by The HDF Group (THG).

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 1998-2006 by the Board of Trustees of the University of Illinois.

All rights reserved.

Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois, Fortner Software, Unidata Program Center (netCDF), The Independent JPEG Group (JPEG), Jean-loup Gailly and Mark Adler (gzip), and Digital Equipment Corporation (DEC).

Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution.
3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change.
4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by The HDF Group and by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign and credit the contributors.
5. Neither the name of The HDF Group, the name of the University, nor the name of any Contributor may be used to endorse or promote products derived from this software without specific prior written permission from THG, the University, or the Contributor, respectively.

**DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP (THG) AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED.** In no event shall THG or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL). The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials: This work was partially produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL.

**DISCLAIMER:** This work was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

### 1.2.3 JPEG Copyright Information

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-1998, Thomas G. Lane. All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions: (1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation. (2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group". (3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA. ansi2knr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2knr.c for full details.) However, since ansi2knr.c is not needed as part of any program generated from the IJG code, this does not limit you more than the foregoing paragraphs do.

The Unix configuration script "configure" was produced with GNU Autoconf. It is copyright by the Free Software Foundation but is freely distributable. The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable.

It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has been removed from the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.) So far as we are aware, there are no patent restrictions on the remaining code.

The IJG distribution formerly included code to read and write GIF files. To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders.

We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

### 1.2.4 PETSc Copyright information

(C) Copyright 1995-2004 University of Chicago

This program discloses material protectable under copyright laws of the United States. Permission to copy and modify this software and its documentation is hereby granted, provided that this notice is retained thereon and on all copies or modifications. The University of Chicago makes no representations as to the suitability and operability of this software for any purpose. It is provided "as is" without express or implied warranty. Permission is hereby granted to use, reproduce, prepare derivative works, and to redistribute to others, so long as this original copyright notice is retained.

Software authors

\* Mathematics and Computer Science Division

\* Argonne National Laboratory

\* Argonne IL 60439 FAX: (630) 252-5986

\* Any questions or comments on the software may be directed to [petsc-maint@mcs.anl.gov](mailto:petsc-maint@mcs.anl.gov).

Argonne National Laboratory with facilities in the state of Illinois, is owned by The United States Government, and operated by the University of Chicago under provision of a contract with the Department of Energy.

DISCLAIMER: THIS PROGRAM WAS PREPARED AS AN ACCOUNT OF WORK SPONSORED BY AN AGENCY OF THE UNITED STATES GOVERNMENT. NEITHER THE UNITED STATES GOVERNMENT NOR ANY AGENCY THEREOF, NOR THE UNIVERSITY OF CHICAGO, NOR ANY OF THEIR EMPLOYEES OR OFFICERS, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS. REFERENCE HEREIN TO ANY SPECIFIC COMMERCIAL PRODUCT, PROCESS, OR SERVICE BY TRADE NAME, TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY THE UNITED STATES GOVERNMENT OR ANY AGENCY THEREOF. THE VIEW AND OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY STATE OR REFLECT THOSE OF THE UNITED STATES GOVERNMENT OR ANY AGENCY THEREOF.

## 2 Introduction

OOPIC Pro is a feature-rich 2-d particle-in-cell (PIC) simulator. It is designed to model plasmas, beams of charged particles, externally generated electric and magnetic fields, and low-to-moderate density neutral gases, using a wide variety of boundary conditions. It includes electrostatic and electromagnetic field solvers, as well as support for x-y (slab) and r-z (cylindrical) geometries. OOPIC Pro provides a convenient and intuitive GUI for use on Microsoft Windows, Mac OS X, and Linux systems, as well as a batch mode to run jobs from the command line.

The OOPIC physics kernel has been used by researchers around the world since 1995 to simulate a wide range of challenging problems. These include plasma display panels, ion implantation, high-power microwave devices, and next-generation particle accelerator concepts. The code is somewhat rare among PIC simulations in its ability to handle ionization of background neutral gasses via electron impact or field-induced tunneling effects. More detailed information is provided by the following papers:

- D.L. Bruhwiler, R.E. Giacone, J.R. Cary, J.P. Verboncoeur, P. Mardahl, E. Esarey, W.P. Leemans and B.A. Shadwick, "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions," *Phys. Rev. ST Accel. Beams* 4, 101302 (2001).
- D.L. Bruhwiler, D.A. Dimitrov, J.R. Cary, E. Esarey, W.P. Leemans and R.E. Giacone, "Particle-in-cell simulations of tunneling ionization effects in plasma-based accelerators," *Physics of Plasmas* 10 (2003), p. 2022.

### 2.1 Objective of This Guide

It is assumed that the reader has no prior experience using OOPIC Pro, but that there is some level of familiarity with particle-in-cell methods and theory. This Guide will show the user how to

- create OOPIC Pro input files that specify a wide variety of physical configurations
- effectively execute the OOPIC Pro application to simulate the specified systems
- use the interactive visual diagnostics of OOPIC Pro to view and understand the results

### 2.2 Organization of This Guide

This guide is divided into several parts, useful for both introduction to the application and as a reference. After Getting Started in Section 3,

- Section 4 describes the OOPIC Pro graphical user interface in detail. This includes instruction on how to start and run the code. The user is instructed on how to create and manipulate diagnostic plots and how to produce output dump files of those plots and of the entire simulation. There is also information about how to create and view movies of the plots.
- Section 5 describes the syntax, the special features and the three block structures of OOPIC input files in detail.
- Section 6 describes the many parameter groups that are used to specify the simulated system. A look through this section will quickly reveal the capabilities of the physics kernel. It also describes the diagnostics available in OOPIC Pro.
- Section 7 provides a number of examples of OOPIC Pro usage to model a variety of devices with different geometries and inputs.

- Section 8 is a list of references containing detailed information about the methods employed in OOPIC Pro.
- Section 9 describes the command-line options available in OOPIC Pro, using OOPIC Pro in parallel, and other advanced features.

## 3 Getting Started

OOPIC Pro provides a GUI that aids the user in understanding the results of the specified PIC simulation. However, correct and effective use of OOPIC Pro requires that the user develop a valid input file, which accurately specifies the physical system to be simulated. This section describes how to install OOPIC Pro and provides an overview of the structure of input files. The GUI is described in Section 4.

### 3.1 Installation

Adding OOPIC Pro to your computer is simple with the cross-platform GUI installer. This can be started by double-clicking the installer icon in Microsoft Windows or Mac OS X. In Linux, the installer can be run by using the command

```
sh ./OOPIC_Pro_2_0_0_eval_installer.bin
```

Note that under Linux, the TERM environment variable must be defined in order for the application to open windows. The existence of TERM is generally assured if OOPIC Pro is being run from a graphical environment.

### 3.2 Evaluation Version of OOPIC Pro

An evaluation version of OOPIC Pro is available for download from [http://www.txcorp.com/products/OOPIC\\_Pro/](http://www.txcorp.com/products/OOPIC_Pro/). The evaluation version is fully functional, but will expire in 30 days. Information on purchasing OOPIC Pro can also be found at this Web site.

## 4 The Graphical User Interface (GUI)

In this section, we describe the GUI that enables the user to interactively start, stop and restart the application, while dynamically viewing field and particle data.

### 4.1 Launching the GUI

Unless otherwise specified, the directions and descriptions that follow apply to all supported platforms. In the discussion below, we will use “OOPIC Pro” as the name of the directory in which the software was installed.

### 4.2 Providing Input Data

A directory called "input" is created with the OOPIC Pro install — this directory contains several example input files. OOPIC Pro input files have a “.inp” extension. Input files can be created and stored anywhere. If they are not stored in OOPIC Pro’s default input directory, then OOPIC Pro will have to be directed to their location by the standard browse operations provided by the Open file dialog box.

When the executable is invoked an Open file dialog window such as that shown in Figure 1 will appear. This dialog allows interactive browsing to the location of the desired input file. The default folder or directory is /OOPIC Pro/input/, where a number of sample input files are kept. Double click on the input file to be loaded.

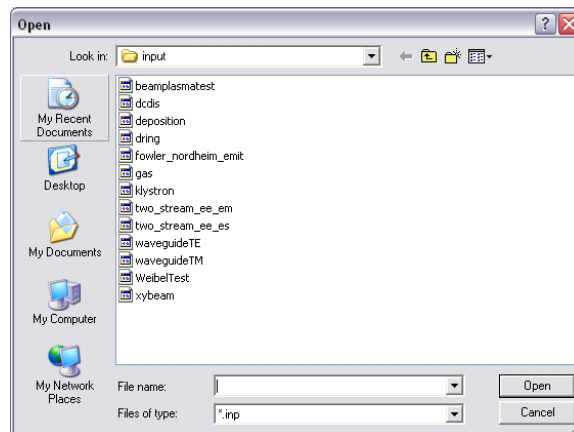


Figure 1: The input file dialog window.

### 4.3 Using the GUI to Control the Simulation

#### 4.3.1 The Control Window

After the input file has been specified, the OOPIC Pro control window will appear. The control window consists of a menu bar, a set of four control buttons and two re-sizable log windows. One window is an output log used for writing diagnostic messages (black text) and the other for error messages (red text). Once the simulation input file has been specified, it will be parsed by OOPIC Pro and loaded. Actions taken by OOPIC Pro based upon contents of the input file will be announced in the output log.

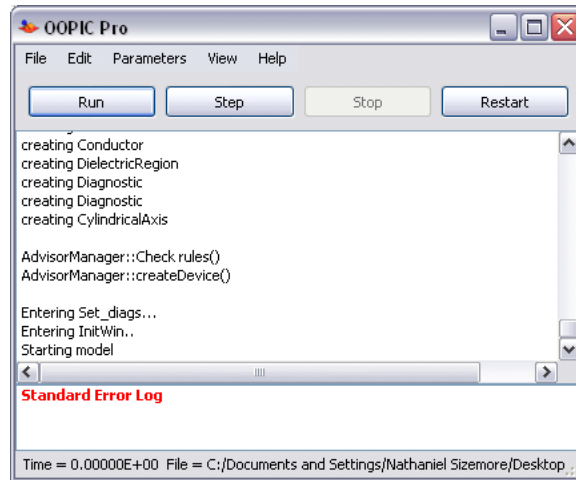


Figure 2: OOPIC Pro Control Window

When the simulation is running the current simulation time is shown on the lower left side of the Control Window border. The name of the current input file can be found in the status bar.

#### 4.3.2 The Control Window Menu Bar

The following is a description of the components of the menu bar and their functions.

**File Menu** The File menu presents the following menu choices.

- Open Input File ... — Open a simulation input file
- Open Dump File ... — Open a previously-saved simulation dump file. Note that if a dump file is opened while a simulation created by a different input file is loaded, the behavior of the application is undefined. Please ensure that the dump file and the input file match.
- Dump current data ... — Save the current state of the simulation ( $E \cdot dl$  and  $B \cdot ds$ , where  $dl$  is the length of the cell segment along the direction of the E field component, and  $da$  is the area of the cell face perpendicular to the B field component) in a designated dump file. The default name will be the base name of the input file, with a ".dmp" or ".h5" extension depending on the format chosen.
- Save image sequence ... — Allows user to save a series of images from the diagnostics which can then be combined into an animation.
- Exit — Quit OOPIC Pro

An input file can be loaded only if the current simulation is stopped. Loading another input file closes the current simulation and starts a new one.

**Edit Menu** The edit menu contains the standard cut, copy, and paste functions — these can be used with the output from both the standard and error windows.

**Parameters Menu** The Parameters menu presents the following items:

- Edit input file ... — Opens a text edit dialog box where the current input file can be viewed or modified.
- Runtime options ... — Opens a dialog box where the user can edit the run parameters of the simulation.

#### **View Menu**

- Diagnostics ... — Opens dialog box where the user can select diagnostics to display
- Log
  - Standard Output — Toggles display of standard output log
  - Standard Error — Toggles display of standard error log

Both the Output Log and the Error Log are present at startup by default.

The output log is located in the upper part of the interior of the main control window. It displays comments on the status of the program, and gives messages as each component of an input file is interpreted and added to the simulation.

The error log is in the lower part of the interior of the main control window. It alerts the user to errors in the simulation that may mean that the simulation should be stopped.

#### **Help Menu**

- OOPIC Pro User's Guide... — Opens the User's Guide using the system's default PDF viewer
- About OOPIC Pro... — Displays copyright and version information

#### **4.3.3 The Control Window Control Buttons**

The four buttons below the menu bar control the execution of the simulation.

- The Run button begins or continues a simulation.
- The Step button advances the simulation one frame. Note that this will not be the same as advancing the simulation one time step if the Number per Update parameter has been changed in the Parameters→Runtime Options... dialog. Changing the Number per Update parameter has the effect of letting the simulation run for multiple time steps between screen/graphics updates. Therefore the [Step] button actually advances the simulation to the time of the next screen update.
- The Stop control stops the simulation.
- The button labeled Restart resumes the simulation from initial conditions.

## 4.4 Displaying Diagnostic Plots

Upon opening an input file in OOPIC Pro, there will be one or more position plots visible, one for each of the species present in the simulation. These plots are examples of default OOPIC Pro diagnostic plots. A full description of the diagnostic plots automatically defined by OOPIC Pro can be found in Section 6.8.1.

OOPIC Pro can also display plots of other diagnostic quantities as well as diagnostics defined by the user (See Section 6.8.3).

To display the plot of any diagnostic (default or user-defined), select the check box next to the diagnostic name in the View→Diagnostics... dialog box.

When you are finished selecting plots, choosing “Done” will close the dialog box.

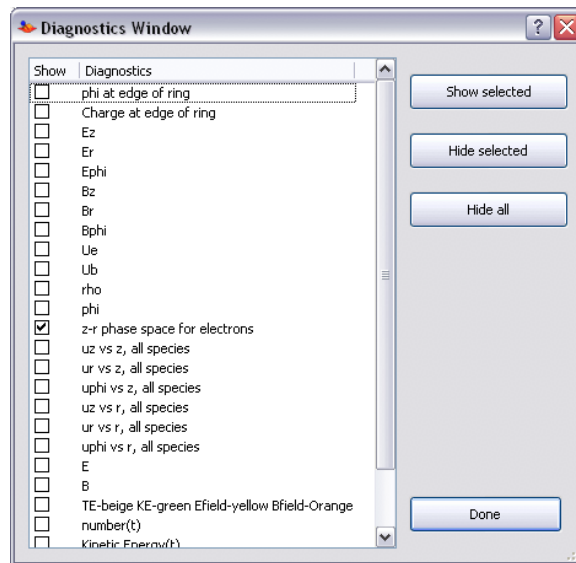


Figure 3: Example Diagnostics Display Selection List

## 4.5 Plot Types

To display diagnostics, OOPIC Pro uses several types of plots. Each is appropriate to the diagnostic quantities being plotted.

### 4.5.1 2-D Particle Plots (configuration space)

A 2-D particle plot (such as Figure 4 shows macroparticle positions within the simulation region. A plot of this type is automatically displayed for each species included in the simulation after the program starts. The axes are either x and y or z and r, depending on the geometry specified for the region in the input file (`Geometry` keyword in the `Grid` block). These plots also show the exterior and interior boundaries defined for the simulation. Each type of boundary (conductor, dielectric, emitter, etc.) is represented by a unique color.

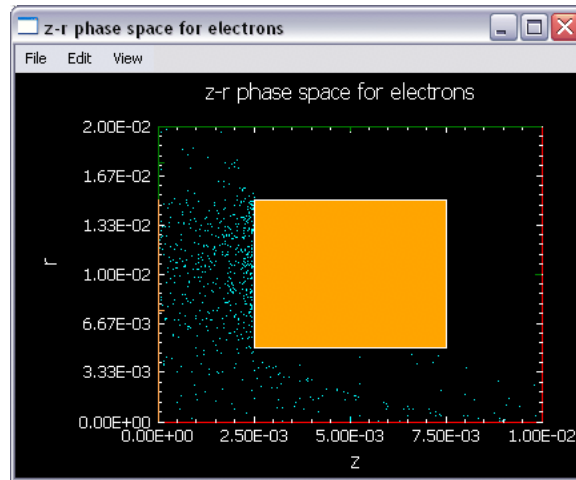


Figure 4: A 2-D Particle Plot

In this plot, the boundary defined to be the BeamEmitter in the input file is drawn in brown. Conductors are drawn in green. The rightmost Conductor boundary does not draw since it corresponds to the limit of the graphics window. This inaccuracy can be fixed by setting a new, larger value for the graphics window maximum via View→ Display Options and entering a new value in the axis maximum value text box.

#### 4.5.2 2-D Phase Space Plots

2-D phase space plots show particle location versus velocity (see Figure 5). Specifically, data for the following plots are automatically collected and the plots are listed in the Diagnostics menu as:

- x1 vs. u1
- x2 vs. u1
- x1 vs. u2
- x2 vs. u2
- x1 vs. u3
- x2 vs. u3

where  $u[i]$  is  $\gamma v[i]$ , with  $v[i]$  being the velocity component of a macroparticle along the  $i$ th dimension.

The convention throughout the OOPIC Pro program and documentation is:

- x1 is the x or z direction, depending on the coordinate system used.
- x2 is either y or r.
- x3 is either z or  $\phi$ , the dimension ignored in the simulation.

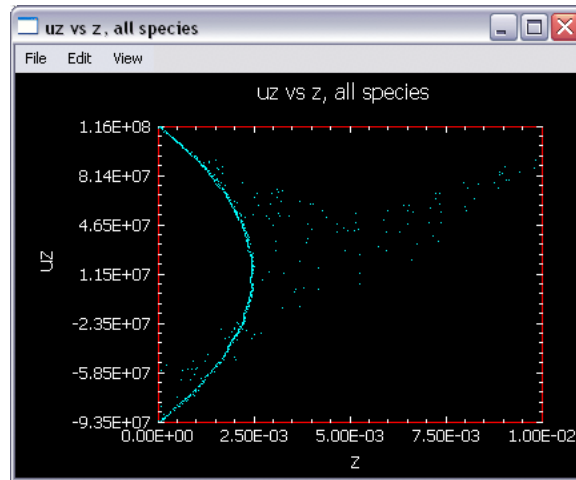


Figure 5: A 2-D Phase Space Plot

#### 4.5.3 2-D Vector Plots

2-D vector plots can be used to show EM field directions and magnitudes. An example can be seen in Figure 6.

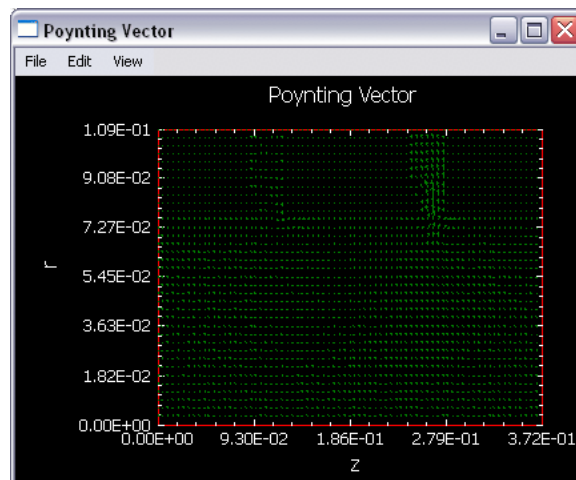


Figure 6: A 2-D Vector Plot

#### 4.5.4 3-D Surface Plots

A 3-D surface plot of the scalar value over the region can be drawn for certain parameters. Examples of diagnostics plotted in this way are the components of the electric and magnetic fields,  $E(x)$ ,  $B(x)$ , etc. Figure 7 is an example of this plot type.

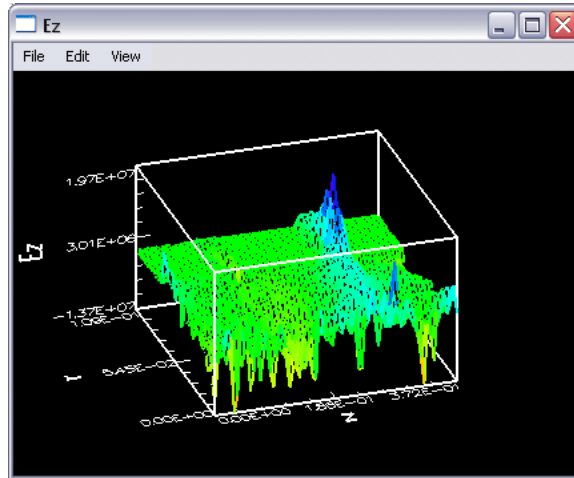


Figure 7: A 3-D surface plot

#### 4.5.5 2-D Line Plots

A 2-D line plot shows a time history of scalar quantities. The default diagnostics which are plotted this way are instantaneous total and kinetic energy of the simulation, average kinetic energy, and the number density of various species. Some plots display more than one quantity.

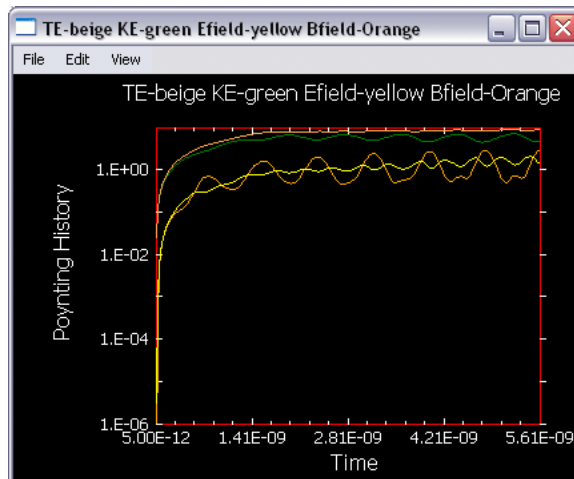


Figure 8: A 2-D Line Plot

## 4.6 Manipulating Diagnostic Plots

### 4.6.1 2-D Plots

Data presented by OOPIC Pro's two-dimensional diagnostics plots can be accessed and manipulated as outlined below. Standard methods of transferring the graphics to other applications are supported, such as clipboard support or exporting to files. In addition, data can be saved in non-graphic file formats that can be accessed by other data visualization software.

**File Menu** Contents of the 2-D plot windows can be saved in graphic or text format, printed, previewed or closed from the File menu.

- Save as... — Saves the contents of the diagnostic window in a variety of graphics formats.
- Dump data... — Saves a description of the E, B, etc. data displayed in a text format which can be analyzed or post-processed. A text dump will create a considerably smaller file than an image file. (Note that the plot dump files are in text format, while simulation dump files are saved in binary format.)
- Print preview... — Print Preview will show what the active plot will look like on a printer chosen from the standard Select Printer dialog box.
- Print... — Brings up a dialog box that allows formatting and printing of the current plot.
- Close — Close will dismiss the plot window. It can be viewed again by selecting it from View→Diagnostics.

**Edit Menu** The Edit menu of the plot window contains a single entry that is used to copy the contents of the plot window to the clipboard (Windows and Linux only). From the clipboard, the window contents can be pasted into other applications that support graphic content.

**View Menu** There are four basic manipulations that can be applied to the 2-d graphics windows that will alter their appearance.

- Crosshair — Choosing Crosshair from the View menu will cause the cursor to be replaced by a crosshair. By positioning the crosshair within the plot window and clicking the left mouse button, x and y values for selected data points can be determined. The x, y data pair for the selected point is displayed at the bottom of the plot window.
- Trace — Choosing Trace from the View menu will cause points in the plot window to not be erased for each graphics update interval (Parameters→ Runtime Options→ Iterations per Plot Update ...). For example, normally if a plot of particle position is displayed against spatial coordinate axes, the entire assembly of particle positions is erased and replotted for each graphics update interval. With the trace feature turned on, the erasure of particle positions does not occur and the trajectory of each particle can be seen as a continuous curve on the plot.
- Zoom In/Zoom Out — It is possible to zoom in to smaller areas of a 2-D plot. Choose Zoom in... from the View menu at the top of the plot window. Position the cursor (an arrow for Windows®, a pointing finger for Linux) at the desired upper left corner of the plot. Click and hold the left mouse button, and drag the cursor down and to the right to the desired lower right corner of the plot and then release the mouse button. The plot will then be redrawn with these new limits.

Selecting Zoom out from the View menu and clicking the mouse with the cursor located anywhere on the plot will result in the plot returning to the previous plot limits. If the plot has been zoomed in more than once, then multiple clicks will be required to return to the original plot limits.

The simulation will be paused during zoom operations, and must be resumed after the user is finished changing the zoom parameters.

- Display Options — The basic appearance of 2-d plots can be controlled through the functions provided on the Display Options menu. Selecting Display Options from the View menu on a 2-d plot produces a selection box with three tab choices available. The tabs are labeled Axes, Title and Tick Marks. (See Figure 9.)

Selecting the Axes tab allows the plot coordinate minimums and maximums to be explicitly set for both axes. Optionally, the axes can be auto-scaled, based on range and the coordinate labels can be written in scientific notation (default) or floating point format. The scale types can also be set to either linear or logarithmic.

Selecting the Titles tab allows the labels attached to the coordinate axes to be changed as well as the color of the label text.

Selecting the Tick Marks tab offers the option of drawing tick marks inside or outside the plot area and allows the color of the tick marks to be set.

The options editor window can also be made to appear by pressing the right mouse button while over the plot.

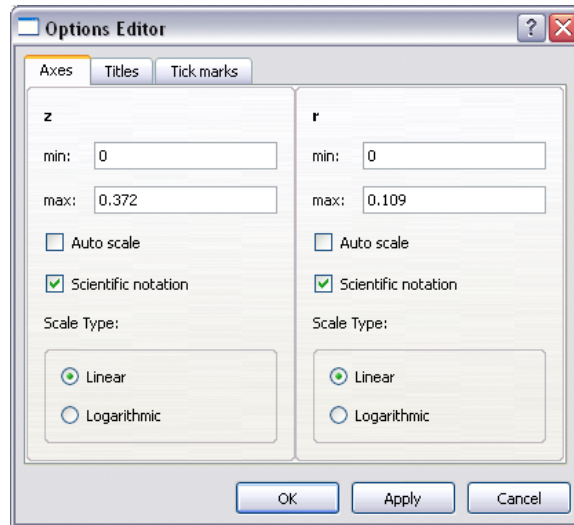


Figure 9: Options Editor window for a 2-d diagnostic plot

#### 4.6.2 3-D plots

Manipulation and management of data presented in OOPIC Pro's 3-D diagnostics plots is very similar to the same processes used for two-dimensional data.

**File Menu** Contents of the 3-D plot windows can be saved in graphic or text format, printed, previewed or closed from the File menu.

- Save as... — Saves the contents of the diagnostic window in a variety of graphics formats.
- Dump data... — Saves a description of the data displayed in a text format which can be analyzed or post-processed. A text dump will create a considerably smaller file than an image file. (Note that the plot dump files are in text format, while simulation dump files are saved in binary format.)
- Print preview... — Print Preview will show what the active plot will look like on a printer chosen from the standard Select Printer dialog box.
- Print... — Brings up a dialog box that allows formatting and printing of the current plot.
- Close — Close will dismiss the plot window. It can be viewed again by selecting it from View→Diagnostics.

**Edit Menu** The Edit menu of the plot window contains a single entry that is used to copy the contents of the plot window to the clipboard (Windows® and Linux only). From the clipboard, the window contents can be pasted into other applications that support graphic content.

**View Menu** The View menu for 3D plots differs significantly from what is offered for 2-d plots. The first four choices in the 3D menu are for turning various graphical characteristics of the plot on or off.

- **Wire Frame** — Wire Frame is on by default and simply shows all six edges of the box that contains the data being plotted as white lines.
- **Shading** — Shading, which is off by default, gives a color to each panel between grid lines depending on the value of the plotted quantity. Reds and yellows denote low values; purples and blues denote high values.
- **Grid** — This option draws lines between grid points. If shading is on, the grid lines are black. If not, they take on colors in the same fashion as shading.
- **Invert B/W** — Invert B/W changes the background of the plot to white. Note that the background is white when printed, regardless of the state of Invert B/W.

**CrossHair** The CrossHair option is disabled for 3-D plots.

## 4.7 Dump Files

The current state of a simulation can be saved in a dump file. There are two choices for the format in which the dump file is saved. The default file format will be HDF5. An OOPIC Pro binary file format is also available.

HDF5 is a common, portable file format using a hierarchical data structure. It can be used by a variety of data analysis and post-processing tools, including IDL.

More information on HDF5 can be found at <http://hdf.ncsa.uiuc.edu/HDF5/>.

### 4.7.1 Saving a Dump File

To save a dump file, select File→ Save dump file... You will be able to choose HDF5 or the OOPIC Pro binary format in the dialog box that appears.

### 4.7.2 Saving a Dump File Periodically

OOPIC Pro can also perform automatic saves of a simulation state at regular intervals. To periodically save simulation status, open the Runtime Options dialog by choosing Parameters→Runtime Options from the main OOPIC Pro window. The number of iterations per data dump can be changed in this dialog, as well as the number of iterations per plot update — note that these can be different values. Each time the dump file saves it will overwrite the previous dump files. The name of the dump file can be specified at the beginning of the simulation by saving a dump file normally as described above. Otherwise, the name of the dump file will assume the default value. To save in binary format, the dump file extension must be specified to be '.dmp', otherwise the save file format will be HDF5 and the filename extension will have to be specified as '.h5'.

### 4.7.3 Loading a dump file

To restore a simulation to the state at which it was when the dump file was saved, choose Open dump file... from the main OOPIC Pro window and select the file.

Note that if a dump file is loaded before OOPIC Pro is configured with the correct input (.inp) file, the simulation behavior is not defined.

## 4.8 Movies

OOPIC Pro has the capability of saving multiple frames of one or more diagnostic plots during a simulation for sequential viewing (i.e. animation) later. To create an OOPIC Pro movie, open the Image Sequence Parameters dialog by choosing File→Save image sequence... from the main OOPIC Pro window.

By default, OOPIC Pro will place the image files in a directory with the same name as the input file. The default image file stub appears in the File Name text input box. The default stub is the input file name followed by an underscore. Users can change the file stub if desired. Users can also choose the format of the movie image files with the “File Type” drop-down menu. The actual image file names will consist of the image file stub, followed by the name of the plot being saved, followed by the frame number, with an image file extension. For example:

```
cavity_z-r_phase_space_for_electrons.00001.png
```

is the first image file in a movie being created for a simulation titled ‘cavity\_z-r\_phase\_space\_for\_electrons’. The image files are saved in the PNG format, as denoted by the file extension.

If more than one movie of the same simulation is to be created, each image should have a unique filename. OOPIC Pro will NOT automatically delete previous image files with the same stub before making a movie. Failing to remove old files with the same stub may result in having extra frames from previous movie.

After choosing an image file stub, the frame rate of the movie should be specified.

Similarly, the maximum number of images can be specified by changing the Images parameter in the dialog box. Its default is ‘1’. If the simulation has multiple graphic plots open, a set of image files will be created from each plot window.

## 4.9 Common problems

- *Program exits immediately upon attempting to open an input file*

It is likely that there has been a parsing error while reading the current input file.

Check the input file for bad syntax — typographic errors and mismatched brackets are common mistakes, and can cause the parser to fail. If the error can’t be found, remember that the only elements necessary for OOPIC Pro to start are the `Grid` and `Control` elements. Comment out all elements but these from the input file. Restore them to the input file one at a time until the error recurs.

- *3D plots aren’t centered in the diagnostic window*

This is a problem with the Mesa graphics library. Forcing the diagnostic window to refresh by changing its size works around this error.

## 5 Overview of Input File Structure and Parameter Groups

OOPIC Pro simulations are defined by a plain text input file. This input file is made up of a series of blocks, delimited by curly braces ('{' and '}'). In each of these blocks, one or more parameters may be set. Some blocks contain blocks themselves. Each line of the file is separated by a carriage return; the use of a semicolon or other character to denote the end of a line will result in a parsing error. The opening and closing brackets must appear alone on separate lines. Comments can be denoted by using "//", similar to C++; note that the OOPIC Pro input file parser will interpret the rest of a line as a comment after seeing "//".

At the highest level, the structure of an input file consists of a required `Description` block, an optional `Variables` block, and a required `Region` block. Section 4.2 explores the basics of the use and definition of information contained in these blocks. More detail is provided for the parameter groups within the `Region` block in Section 5.3.

### 5.1 Description Block

The `Description` block is a required part of the input file, and gives a user a place to describe a broad overview of the geometry of the simulation, the particle species involved, interesting properties that are being investigated in the simulation, etc.. The contents of this block are passed directly through the parser without interpretation, so it is not necessary to denote comments in this block by "//". The name of the `Description` block is usually chosen to be the same as the name of the input file itself, but this is not a requirement. A simple example of this block might be

```

Foo
{
  This file is an example of an OOPIC Pro Description block.
  Comments about the geometry, particles, literature references, etc.
  can go here.
}

```

### 5.2 Variables Block

An optional `Variables` block, positioned immediately before the `Region` block, can contain user-defined symbolic variables that can be referenced later in the body of the input file. Variables are defined here and assigned numeric values with the "=" operator. Subsequent reference can be made to the variable on the right-hand side of a parameter, and a numeric equivalent will be substituted in its place by the input parser before evaluation. Using variables to describe properties such as the number of cells in the simulation, physical properties, etc. both makes the input file easier to read, and allows the simulation to be changed quickly by keeping parameters used by multiple blocks in a single location.

An example of a `Variables` block might look like the following:

```

Variables
{
  Jmax = 200      // number of cells in x1 direction
  Kmax = 120      // number of cells in x2 direction
  khalf = KMAX/2 // middle cell in x2
}

```

### 5.3 Region Block

Parameter group definitions occur within the `Region` block. Numeric values (or their symbolic equivalents defined in the `Variables` block) for all parameters of the simulation will be found in this block. The curly bracket denoting the end of the `Region` block is always the last line the input file.

Input file parameter groups that are specified within the `Region` block can be divided into six different categories. These categories correspond to the different types of physical entities that OOPIC Pro can model.

This section briefly describes these parameter groups. Detailed information about constituent parameters, how they are used and the values they can be assigned is provided below in Section 6 of this guide.

Parameter group categories are:

1. Grid and Control — the parameter groups in this category must appear in the input parameter file.
  - (a) Grid — Specifies the physical dimensions of the simulated region, whether Cartesian or cylindrical geometry is to be used, the details of the grid or mesh, and other related parameters.
  - (b) Control — Specifies the time step for the simulation, external electric and magnetic fields, whether to use electrostatic or electromagnetic field solve, a seed value for the random number generator and other global parameters.
2. Particle Creation — this parameter group is used to specify how and when particles enter the simulation. If the addition occurs at time zero in the simulation, then this initial presence will need distribution parameters supplied for position, velocity, etc. in the `Load` segment for each `Species`.
  - (a) Species — Specifies the characteristics of a particle type present in the simulation. One `Species` block is included for each type of particle. Different groups of particles are distinguished by their names in the `Species` blocks. This is useful, *e.g.*, in tracking electrons that derive from different sources.
  - (b) MCC — Monte Carlo collision parameters used to model collisions of particles with a background gas as well as the resulting ionization of the gas. Multiple gas types can be supported, each with its own `MCC` block.
  - (c) Load — Parameters specifying the initial spatial and Maxwellian velocity distributions of a named particle species.
  - (d) `VarWeightLoad` — Same as `Load`, but uses variable weighting of the macro-particles in order to keep the macro-particle number density uniform across a cylindrical grid. This is achieved by linearly increasing the weighting factor of the macro-particles with the radius.
  - (e) `PlasmaSource` — defines a rectangular region in which a plasma is generated at a constant rate, as well as the initial placement and velocity profile of the plasma constituents.

Particles can also be created by emission from a surface. This is discussed in the subsection on particle emitters below.

3. Boundary Conditions — OOPIC Pro supports the definition of a number of physical boundary types as listed below. Most of these types require parameters to specify geometric or time-dependent properties. For some boundary conditions, complex geometries can be specified by defining multiple `Segments`. Detailed information for each boundary condition type and use of `Segment` blocks is given in Section 6.4.1, along with examples.
  - (a) Dielectric — Specifies a dielectric material with arbitrary index of refraction. For electrostatic simulations, charge can accumulate as particles hit the surface. Only one `Segment` specifier allowed.
  - (b) Conductor — Uses same parameters as Dielectric, but imposes perfect conducting BC's on fields, and multiple `Segment` specifiers are allowed. Any charge or current from particles hitting the boundary is effectively conducted away in zero time.

- (c) Equipotential — Behaves like a perfect conductor, which is grounded to a specified potential, which may vary in time.
- (d) Polarizer — Applies conducting BC's to the fields, but allows a fraction of incident particles to pass through, based on a specified transmissivity.
- (e) Secondary — Specifies how secondary electrons are to be produced at a boundary. Production can be limited to incidence of a designated species.
- (f) Secondary2 — A Vaughan-based model, which includes energy and angular dependence and a full emission spectrum, including reflected and scattered primaries, as well as true secondaries.
- (g) Secondary3 — A secondary electron emission model in which the electron-induced secondary emission is handled using Furman and Pivi model; ion-induced secondary emission is handled using H. Rothard model
- (h) DielectricRegion — Same parameters as Dielectric, but includes coordinates to specify a rectangular region rather than a simple linear segment.
- (i) DielectricTriangle — Same parameters as Dielectric and DielectricRegion but includes coordinates to specify the vertices of a right triangle.
- (j) CurrentRegion — Describes the magnitude and spatial distribution of the current flowing within a specified region of the grid.
- (k) Iloop — Causes a loop of current to be included in the simulation. It is useful as an external source.
- (l) CylindricalAxis — This special boundary condition is necessary for an r-z grid (cylindrical geometry), if the line  $r=0$  is included in the simulation.

#### 4. Ports

- (a) ExitPort — A boundary where electromagnetic waves can exit the grid, with minimal reflection.
- (b) Gap — A wave type boundary condition, where a sinusoidal field variation is applied.
- (c) PortGauss — Launches an electromagnetic wave pulse with a transverse Gaussian profile and a specified longitudinal profile. This Port can only be used in Cartesian geometry.

#### 5. Particle Emitters

- (a) BeamEmitter — Produces a beam of macro-particles of the specified Species, with the specified particle current. Current can vary in time and space. Control is provided over beam temperature. Perfect conducting BC's are applied to the fields.
- (b) EmitPort — Same as BeamEmitter, but it does not apply any boundary conditions to the fields.
- (c) VarWeightBeamEmitter — Same as BeamEmitter, but it uses variable weighting of the macro-particles in order to keep the macro-particle number density uniform across a cylindrical grid. This is achieved by linearly increasing the weighting factor of the macro-particles with the radius.
- (d) FowlerNordheimEmitter — Models field emission of electrons from a surface, according to the Fowler-Nordheim theory.

- 6. User-Defined Diagnostics — There are a number of computed quantities that can be graphically monitored during the course of a simulation.

## 5.4 Example Input File

The following example shows the basic structure of an OOPIC Pro input file. Information regarding the meaning and use of data defined in the parameter groups can be found in Section 6 below.

```

ExampleInputFile
{
  // The Description block. The name and contents of this block have no effect
  // on program operation. This is an example of an input file.
}
// Variables defined in this block can be referred to elsewhere in the input file
Variables
{
  x1max = .1
  x2max = .02
  Jmax = 16
  Kmax = 16
  Cartesian = 1 // specify Cartesian geometry
}
// Region block. All parameter group blocks must be within the Region block
Region
{
  // Grid block - used to specify Region size and mesh parameters
  Grid
  {
    J = Jmax
    x1s = 0.0
    x1f = x1max
    n1 = 1.0
    K = Kmax
    x2s = 0.0
    x2f = x2max
    n2 = 1.0
    Geometry = Cartesian
  }
  // Control block
  Control
  {
    dt = 1.0E-11 // specifies time step in seconds
    ElectrostaticFlag = 1 // specifies electrostatic field solve
  }
  // Species block
  {
    name = electrons // name of the particle species
    m = 9.11E-31 // physical mass of the particles
    q = -1.6E-19 // physical charge of the particles
  }
  BeamEmitter // Specifies boundary which emits a beam of particles
  {
    j1 = 0 // lower endpoint in 1st dimension
    k1 = 0 // lower endpoint in 2nd dimension
    j2 = 0 // if j2=j1, then emitter is parallel to 2nd axis
    k2 = Kmax/4 // upper endpoint in 2nd dimension
    normal = 1 // direction of emitter
    speciesName = electrons // must be defined above in Species block
    I = 10 // Emission current
    np2c = 1.0E7 // numerical weight of the macro-particles
    // (# of physical ptcls per computational ptcl)
    v1drift = 1e7 // speed of particles along 1st dimension in m/s
  }
}

```

```
}  
Conductor // perfect conducting boundary  
{  
  j1 = 0 // This is the boundary condition for the left side of the region  
  k1 = Kmax/4  
  j2 = 0  
  k2 = Kmax  
  normal = 1  
}  
Conductor  
{  
  j1 = Jmax // This is the boundary condition for the right side of the region  
  k1 = Kmax  
  j2 = Jmax  
  k2 = 0  
  normal = -1  
}  
Conductor  
{  
  j1 = 0 // This is the boundary condition for  
  k1 = 0 // the bottom side of the region  
  j2 = Jmax  
  k2 = 0  
  normal = 1  
}  
} // End of the Region Block
```

## 6 Blocks and Parameters in OOPIC Pro Input Files

OOPIC Pro simulations vary from simple to complex, but they are all built from the same components and adhere to the same syntax. This section provides a reference for all blocks available for OOPIC Pro, their parameters, and examples of use.

### 6.1 Syntax, Operators, and Built-in Functions

The syntax rules implemented by the OOPIC Pro input file parser provide for a variety of data types, a number of arithmetic operators, symbolic constants and special functions.

#### 6.1.1 Data types

OOPIC Pro input file parameters can be one of four types: integer, string, scalar, or flag. These are described in the following table.

Table 1: OOPIC Pro Data Types

Data Type	Usage
int	integer
string	Any set of characters with no whitespace
scalar	floating point single or double precision number in either decimal or scientific notation
flag	Binary flag — 0 or 1

#### 6.1.2 Floating Point Notation

Both decimal and scientific notation are allowed.

All of the following are understood by the parser and are equivalent.

- 0.03
- .03
- 3e-2
- 3E-2
- 3e-02

### 6.1.3 Operators

Standard arithmetic operators are understood by OOPIC Pro and can be used in input files. These are described in the following table.

Table 2: Operators Defined in OOPIC Pro

Operator	Description
=	assignment
-	subtraction, negation (unary minus)
+	addition
*	multiplication
/	division
%	modulus
^	exponentiation

### 6.1.4 Constants

The constants PI and e are built-in, and do not need to be defined in an OOPIC Pro input file. Note that the input file parser is case-sensitive — “PI” must be in all capitals, and the number “e” must be lowercase.

Table 3: Constants Defined in OOPIC Pro

Operator	Value
PI	3.141592654
e	2.18281828

### 6.1.5 Functions

The OOPIC Pro parser implements the following set of mathematical functions:

Table 4: OOPIC Pro Mathematical Functions

Function	Description
sin()	sine of argument
cos()	cosine of argument
atan()	arctangent of argument
ln()	natural logarithm of argument
exp()	natural exponential of argument
sqrt()	square root of argument
pow(x,y)	x to the yth power
pulse(t, tdelay, trise, tpulse, tfall)	finite pulse
step(x)	returns 0 if $x < 0$ and returns 1 otherwise
ramp(x)	returns 0 if $x < 0$ and returns x otherwise

The pulse function is a special function created for users that allow boundaries that exhibit time-dependent behavior to be defined. For example, `Equipotential` boundaries can have a time-dependent voltage, and `BeamEmitters` can have time-dependent currents.

### 6.1.6 xtFlag — Default Settings

`xtFlag` is a boundary parameter that applies to all types of boundaries implemented in OOPIC Pro. This is a generic boundary parameter, meaning that it can be used in the specification of all OOPIC Pro boundary types. These include conductors, equipotentials, emitters and any other parameter group belonging to the boundary class. The default value for `xtFlag` is 0. This implements time-dependence of the form:

$$F(t) = Envelope(t) \cdot Sinusoid(t)$$

where  $t$  is the simulation time.

The envelope function is a piecewise linear function whose parameters specify a function like that shown in Figure 6.1.6. The envelope is completely parameterized by specifying the values for `a0`, `a1`, `tdelay`, `trise`, `tpulse` and `tfall`.



Figure 10: Diagram of Time Dependent Envelope Function

The sinusoid is parameterized by values for: phase, frequency,  $A$  and  $C$  in the function:

$$F(t) = A \cdot \sin(2 \cdot \pi \cdot frequency \cdot t + phase) + C$$

An example of the use `xtFlag` can be found in the input file `beamplasmatetest.inp`.

## 6.2 Overall Simulation Parameters

### 6.2.1 Grid

The `Grid` element is used to specify the size, shape, geometry and other characteristics of the numerical grid superimposed upon the simulated system. It establishes a correspondence between the numerical grid and the geometrical parameters of the system. This block is required for all OOPIC Pro input files.

The grid parameters listed here without default values are required to be specified in the input file.

Note: In this guide, the names of the coordinate axes are referred to as `x1`, `x2` and `x3`. If the `Geometry` flag in this input element is set to a value of 1, then the simulation is computed for a Cartesian coordinate system where `x1` corresponds to the  $x$  direction, `x2` to  $y$ , and `x3` to  $z$ . If `Geometry` is set to 0, then the simulation is computed for a right cylindrical coordinate system where `x1` corresponds to  $z$ , `x2` corresponds to  $r$ , and `x3` corresponds to  $\phi$ . In both coordinate systems, `x3` is the coordinate of symmetry and can be ignored.

Table 5: Grid Parameters

Parameters	Default value	Data type	Units	Description
J	None	int		Number of cells in the x1 direction.
K	None	int		Number of cells in the x2 direction.
x1s	None	scalar	m	Lower coordinate in the x1 direction.
x1f	None	scalar	m	Upper coordinate in the x1 direction.
x2s	None	scalar	m	Lower coordinate in the x2 direction.
x2f	None	scalar	m	Upper coordinate in the x2 direction.
Geometry	0	flag		Cylindrical (0) or Cartesian (1) geometry
PeriodicFlagX1	0	flag		Non-periodic (0) or periodic (1) boundary conditions in the first direction. (Applicable to 1-d problems)
PeriodicFlagX2	0	flag		Non-periodic (0) or periodic (1) boundary conditions in the second direction
n1	1	scalar		Scaling parameter for non-uniform grids in the x or r direction. The mapping is defined as $x_j = x1s + (x1f - x1s)(j/J)^{n1}$ , $0 \leq j \leq J$
n2	1	scalar		Scaling parameter for non-uniform grids in the y or theta direction. The mapping is defined as $x_j = x1s + (x1f - x1s)(j/J)^{n2}$ , $0 \leq j \leq J$

Here is an input block example showing the use of Grid:

```

Grid
{
  // No geometry specified, so using default of 0 (cylindrical)
  J = 30      // simulation has 30 cells in r direction
  x1s = 0.0  // start of sim in r is 0 meters
  x1f = 0.05 // end of sim region in r is 0.05 meters
  K = 30      // sim has 30 cells in theta direction
  x2s = 0.0  // start of sim in theta is 0 meters
  x2f = 0.01 // end of sim in theta is 0.01 meters
}

```

## 6.2.2 Control

The Control element is used to specify the time step for the simulation, initial static E and B field parameters, a default seed value for random number generator and other general parameters that apply to the overall simulation.

Table 6: Control Element Parameters

Parameters	Default value	Data type	Units	Description
dt	1E-12	scalar	s	Simulation time step
emdamping	0	scalar		Damping value for high frequency noise. $0 < \text{emdamping} < 1$
movingWindow	0	int		Set to 1 to enable "moving window" feature.
shiftDelayTime	0	scalar	s	Length of time before the moving window is activated.
gasOffTime	-1.0	scalar	s	Time at which to set the neutral gas density to zero.

Frandsseed	0	int		Seed for random number generator.
B01	0.0	scalar	Tesla	Uniform static magnetic field in x1 direction.
B02	0.0	scalar	Tesla	Uniform static magnetic field in x2 direction.
B03	0.0	scalar	Tesla	Uniform static magnetic field in x3 direction.
B01analytic	"0"	string	Tesla	x1 component of the static magnetic field as a function of x1 and x2
B02analytic	"0"	string	Tesla	x2 component of the static magnetic field as a function of x1 and x2
B03analytic	"0"	string	Tesla	x3 component of the static magnetic field as a function of x1 and x2
Bf	NULL	string		Name of the file that contains the values of the magnetic field at each node.
E1init	"0"	string	V/m	x1 component of an initial electric field given as an analytic function of x1 and x2
E2init	"0"	string	V/m	x2 component of an initial electric field given as an analytic function of x1 and x2
E3init	"0"	string	V/m	x3 component of an initial electric field given as an analytic function of x1 and x2
B1init	"0"	string	Tesla	x1 component of an initial magnetic field given as an analytic function of x1 and x2
B2init	"0"	string	Tesla	x2 component of an initial magnetic field given as an analytic function of x1 and x2
B3init	"0"	string	Tesla	x3 component of an initial magnetic field given as an analytic function of x1 and x2
j1BeamDump	-1	int		Grid coordinate of left side of beam dump (magnetic field). -1 => <i>donotuse</i>
j2BeamDump	-1	int		Grid coordinate of right side of beam dump (magnetic field). -1 => <i>donotuse</i>
MarderIter	0	int		Number of iterations of Marder correction for non-charge-conserving particle current.
MarderParameter	0.5	scalar		Relaxation parameter for Marder correction.
ElectrostaticFlag	0	flag		Determines field solver. See below.
PetscKSPType	7	flag		elect the type of PETSc KSP solver used in the Poisson problem
initPoissonSolve	1	flag		Do an initial electrostatic Poisson solve at t=0.
CurrentWeighting	0	flag		Determines current weighting for EM model: 0=charge conserving (Nearest grid point) 1=bilinear in current (may require divergence correction).
DivergenceCleanFlag	0	flag		Determines whether a divergence correction will be applied to the electric field: 0 = no, 1 = yes
NonRelativisticFlag	0	flag		1=NonRelativistic particle push; 0=Relativistic
nSmoothing	0	int		Number of binomial smoothing passes (0=none)
InfiniteBFlag	0	flag		1=NonRelativistic, infinite B1; 0=normal, relativistic particle push
np2cFactor	1	scalar		Increases the numerical weighting (np2c) of particles read in from the dump file.
particleLimit	1e8	scalar		max number of particles in the simulation. Halves the number of particles when exceeded.
SynchRadiationFlag	0	flag		1 = Synchrotron radiation model for particles tracking in electromagnetic simulations; 0 = No synchrotron radiation model.
presidue	1e-3	scalar		Specifies a residue for the Poisson Solver.

A simple example of a `Control` block would be:

```
Control
{
  dt = 3.0E-10           // time step in seconds
  ElectrostaticFlag = 1 // specifies electrostatic simulation field solver
}
```

The `Control` block also contains some special features, such as use of moving windows, external magnetic fields, or calculating radiation from plasmas. The use of these options is detailed below.

**movingWindow** This will move fields and particles in the simulation one cell to the left at the speed of light with each time step. Boundaries are NOT moved. Additionally a load with the name "shiftLoad" will be loaded whenever a shift occurs. An example of this feature can be found in `beamplasmatest.inp` or `gaussTest.inp`.

**ElectroStaticFlag** OOPIC Pro provides both electromagnetic and electrostatic representation of the self-consistent fields. For the electromagnetic case, particle currents on the grid are used to self-consistently drive the full set of Maxwell equations for the electric (E) and magnetic (B) fields. The electrostatic representation, if chosen by using the `ElectrostaticFlag` parameter, is a strong approximation. This is very useful and physically correct in certain situations — particle currents are ignored, while particle charges on the grid are used to solve Poisson's equation for the electric field. An external magnetic field can be added to an electrostatic simulation (using the `Bf` parameter; see below), but magnetic fields will never arise dynamically. However, the electrostatic approximation to Maxwell's equations requires that the following assumptions be satisfied:

1. The light-crossing time for the domain of interest must be much shorter than any other physical time scale in the problem.
2. There must not be any propagating electromagnetic waves.
3. Any magnetic fields that would be generated by particle currents must be negligible.
4. Particle motion is non-relativistic.

If these assumptions cannot be met or are incorrect for a given simulation, it is recommended that the simulation be run using the electromagnetic model instead.

The `ElectrostaticFlag` parameter, as described in the table above, allows the user to select among various algorithms for solving Poisson's equation to get the electrostatic potential ( $\phi$ ) from the charge density ( $\rho$ ). One then finite-differences  $\phi$  to get the electric field components. When finite-differencing the Poisson equation  $\text{del-square-dot-phi} = \rho / \text{epsilon0}$  you get a large matrix equation, which you can try to solve in various ways. The `ElectrostaticFlag` allows the user to select the type of solver OOPIC Pro uses.

- If you set `ElectrostaticFlag=0` (the default), then the fields on the grid will be determined by a full update of Maxwell's equations, driven by the current deposited by particles on the grid. In such simulations, you can speed up the particle push for nonrelativistic particles by setting `NonRelativisticFlag=1`, because the code doesn't bother to calculate the expensive gamma factor of special relativity. However, this choice will lead to incorrect particle motion if the particles are actually relativistic.

- ADI stands for "alternating direction implicit". DADI stands for "dynamic" ADI; this method is selected by setting `ElectrostaticFlag = 1`.
- Flag 2, "conjugate gradient", is a standard technique for solving matrix equations.
- Flag 3, GMRES, stands for the "Generalized Minimal Residual" method.
- "Multigrid", option 4, is a name for a class of techniques where you first solve the problem on a very coarse mesh, then use that solution as a starting point for solving on a finer mesh, etc. until you get the solution on the actual mesh you are using.
- Setting `ElectrostaticFlag = 5` selects Periodic DADI. This is a variant of DADI, specialized to periodic boundary conditions.
- PETSc is a massively parallel software library for doing linear algebra. OOPIC Pro can be directed to use this package for solving electrostatic simulations by setting `ElectrostaticFlag = 6`. PETSc is under active development, primarily at Argonne National Lab.

For details on both ADI and conjugate gradient techniques, the Numerical Recipes chapter on Partial Differential Equations is a good start. More information on PETSc can be found at <http://www-unix.mcs.anl.gov/petsc/petsc-as/>.

**PetscKSPType** A flag is available to select the type of PETSc KSP solver used in the Poisson problem. This flag is defined as "PetcKSPType" (default value is 7, CGS type) and the list of KSP solver options available are:

- 1 - Richardson
- 2 - Chebychev
- 3 - CG
- 4 - GMRES
- 5 - TCQMR
- 6 - BCGS
- 7 - CGS
- 8 - BICG

**Plasma Radiation** OOPIC Pro can compute the impurity (or plasma) radiated power values using the TxPhysics library. The plasma radiation calculation can be set up when the user specifies the "PlasmaRadiationFlag" as 1 under the control block of OOPIC Pro input file (the default value of `PlasmaRadiationFlag` is set to 0 which means no plasma radiation calculations).

The TxPhysics radiation routines adopt Coronal model (from the numerical algorithms given by D. Mosher, "Coronal equilibrium of high atomic number plasmas", Phys. Rev. A 10, 2330 - 2335 (1974)) for determining the total continuum and line radiated power values of the plasma including Brehmstrahlung and recombination.

The total continuum radiation power scales linearly as the plasma density (both electron and ion densities) and as the square root of electron temperature(T) as given below:

$$P_{cont} = A_{cont} n_e n_i T^{1/2}$$

Here  $A_{cont}$  is a temperature-dependent factor.

The line radiation scales linearly as the plasma density (both electron and ion densities) and as the inverse square root of temperature, multiplied by a temperature-dependent factor  $A_{line}$ :

$$P_{line} = A_{line} n_e n_i T^{-1/2}$$

Also a total radiated power is calculated by summing the continuum and line radiated power values. It is given by

$$P_{total} = P_{cont} + P_{line}$$

Following diagnostic options are available to analyze the plasma radiation during a simulation:

- Time history results of the continuum radiated power
- Time history results of the line radiated power
- Time history results of the total radiated power
- Contour plot of distribution results of the continuum radiated power in the entire simulation domain
- Contour plot of distribution results of the line radiated power in the entire simulation domain
- Contour plot of distribution results of the total radiated power in the entire simulation domain
- Contour plot of distribution results of electron temperature (in eV) in the entire simulation domain
- Contour plot of distribution results of ion temperature (in eV) in the entire simulation domain

All radiated power values are given in  $W/m^3$ .

For an example of using the `PlasmaRadiationFlag`, see the input file `plasmaradiation.inp`.

**Bf — Specifying External Magnetic Fields** External magnetic fields can be defined in OOPIC Pro by creating a text file describing the field. This allows users to create a static B field in the simulation. The Bf parameter is used in the Control block to give OOPIC Pro the location of the text file describing the field. This is a filename which contains the values of the magnetic field at each node. For example,

```
Control
{
  dt = 1.0E-11
  ElectrostaticFlag = 1
  Bf = /Users/johnDoe/Bf.dat
}
```

The file itself is a whitespace-delimited text file with five columns; either

```
x y Bx By Bz
```

for Cartesian geometry, or

```
z r Bz Br Btheta
```

for cylindrical coordinates. The values for the magnetic fields are specified in units of Gauss. The numbers in this data file are used to populate an array internally; as such, the second column (y or r) is iterated completely before iterating the first. For example,

```
.
.
.
-1.000000 1.800000 -148.931600 -123.844100 0.000000
-1.000000 1.840000 -149.426200 -117.330600 0.000000
-1.000000 1.880000 -149.719800 -111.134800 0.000000
-1.000000 1.920000 -149.807700 -105.236500 0.000000
-1.000000 1.960000 -149.789900 -99.592880 0.000000
-1.000000 2.000000 -149.649000 -94.242770 0.000000
-0.960000 0.000000 -58.116420 148.210300 0.000000
-0.960000 0.040000 -47.837470 152.049500 0.000000
-0.960000 0.080000 -37.104070 155.124800 0.000000
-0.960000 0.120000 -25.930190 157.350100 0.000000
.
.
.
```

This data is usually generated in an outside program, such as IDL, MATLAB, or similar package.

**FreezeFields — Specifying Constant Electric Fields** Users can specify a constantly applied E-field (without having a Poisson solve to account for the space charge effects) by using the `FreezeField` parameter in the `Control` block. To include a constant E field in an OOPIC Pro simulation, set `FreezeFields = 1` in the `Control` block, along with the parameters `E1init`, `E2init`, and `E3init` that specify the E field in the corresponding direction. The initialization parameters can be set with either fixed values or analytical expressions.

## 6.3 Species blocks and Particle Creation

The following elements of an input file allow for the specification of initial or dynamic particle distributions as well as the types and properties of particles created in this way.

### 6.3.1 Species

The `Species` block specifies the characteristics of a macroparticle type modeled by the simulation. A macroparticle is an assemblage of `np2c` (described in the `Load` element) individual, physical particles of the same type that are modeled as a single unit. Each type of macroparticle requires its own, unique `Species` element. For simulations that model multiple types of particles (for example, electrons and ions), there will be more than one `Species` block within the input file.

Identical particle types can be created within OOPIC Pro by different sources. For example, electrons can be injected by a beam emitter and spawned by some interaction process such as secondary scattering. In this situation, the definition of a `Species` element for each source will group the created particles independently even though they have the same physical properties. Regardless of their grouping, they are treated the same by the physics kernel.

Table 7: Species Parameters

Parameters	Default value	Data type	Units	Description
name	NULL	string		Unique name for this species.
q	-1.60 E-19	scalar	C	Charge per physical particle of this species.
m	9.11 E-31	scalar	kg	Mass per physical particle of this species.
collisionModel	None	string		Model to use for collisions of this species: 0: none 1: electron 2: ion 3: test
threshold	0	scalar	eV	If the total energy of the particle drops below this value then it is transferred to the BoltzSpecies.
particleLimit	1e8	int		Maximum number of particles of this species in the simulation. If this value is exceeded then the number of particles is halved.
subcycle	1	int		Number of field advances per particle advance.
rmsDiagnosticsFlag	0	int		Set to 1 to collect time history plots of RMS values for beam size and velocity.

Input Block Example:

```
Species
{
  name = neon
  m = 3.376e-26
  q = 1.6E-19
  subcycle = 20
  collisionModel=2
}
```

### 6.3.2 MCC (Monte Carlo Collisions)

This element specifies the Monte Carlo collision parameters used to model collisions of particles with background gases that might result in the ionization of the background gases and the production of a pre-defined species of particle.

Table 8: Monte Carlo Collision Parameters

Parameters	Default value	Data type	Units	Description
gas	NULL	string		Background gas species. Possible values are: H, He, Li, Ar, Ne or Xe.
pressure	0	scalar	Torr	Uniform pressure for this gas.
temperature	0.025	scalar	eV	Uniform gas temperature.
eSpecies	NULL	string		Electron species to create in ionization collisions.
iSpecies	NULL	string		Ion species to be created in ionization collisions.
iSpeciesPlusPlus	NULL	string		Ion species to be created in ionization collisions.
ionzFraction_i	1	int		Number of ions to create per ionization event.
ecxFactor	1	scalar		Scale all electron cross sections for this gas by ecxFactor
icxFactor	1	scalar		Scale all ion cross sections for this gas by icxFactor.
relativisticMCC	0	int		Flag: 0=Original MCC model 1=Relativistic MCC model
x1MinMKS	-1.	scalar	m	minimum x1 position defining neutral gas region
x1MaxMKS	-1.	scalar	m	maximum x1 position defining neutral gas region
x2MinMKS	-1.	scalar	m	minimum x2 position defining neutral gas region
x2MaxMKS	-1.	scalar	m	maximum x2 position defining neutral gas region
analyticF	0	string		Analytic function of x1, x2 describing the neutral gas density. The default evaluation of the function will return 0.0
collisionFlag	1	int		Enable (1) or disable (0) Monte Carlo collisions
tunnelingIonizationFlag	0	int		Enable (1) or disable (0) the tunneling ionization. Supported gas types for this model are: H, He, Li, and Cs.
ETIPolarizationFlag	0	int		Specifies E field polarization (0 = linear, 1 = circular) in the calculation of tunneling ionization probability. Requires tunneling ionization to be enabled if circular polarization is to be used.
EfieldFrequency	-1.	scalar		Frequency of the external alternating E field
TI_numMacroParticlesPerCell	10	scalar		Number of macro particles in a cell due to tunneling ionization. Requires tunneling ionization to be enabled.

Input Block Example:

```
MCC
{
  gas = Ne           // name of the neutral gas
  pressure = 0.045   // pressure of the neutral gas
  eSpecies = electrons // electron species produced by collisions
  iSpecies = neon     // ion species produced by collisions
}
```

The example input file `gas.inp` contains a complete simulation using MCC blocks.

OOPIC Pro considers the following default particle-particle collision types for the various background gas types for Monte Carlo Collisions:

- Argon: electron - Ar neutral = Elastic, Excitation and Ionization
- Argon ion - Ar neutral = Elastic and Charge exchange
- Neon: electron - Ne neutral = Elastic, Excitation, and Ionization
- Ne ion - Ne neutral = Charge exchange
- Xenon: electron - Xe neutral = Elastic, Excitation, and Ionization
- Xenon ion - Xe neutral = Charge exchange
- Hydrogen: e-H = Ionization
- Helium: electron - He neutral = Elastic, Excitation, and Ionization
- Helium ion - He neutral = Elastic and Charge exchange
- Lithium: electron - Li neutral = Elastic and Ionization
- Nitrogen: electron - N neutral = Elastic and Ionization

Note that tunneling ionization (using `tunnelingIonizationFlag`) supports a limited set of background gases. Supported gas types for this model are H, He, Li, and Cs.

### 6.3.3 MCC (Monte Carlo Collisions) with User-Defined Cross Section Data

MCC blocks also allow users to specify their own collision cross section data to model specific electron-neutral and ion-neutral collisions. In an MCC block, users can specify the location of an external file that contains the collision cross section details.

The example below illustrates this new feature in the MCC block.

```
MCC
{
  gas = Ar
  pressure = 1.e-3
  eSpecies = electrons
```

```

iSpecies = argonions
NElectronCollisions = 3
ElecCollXSectionFile = electronCSection.dat
NIonCollisions = 2
IonCollXSectionFile = ionCSection.dat
}

```

To enable user-defined cross sections, the following parameters are used in the MCC block.

**NElectronCollisions (default = 0)** This parameter defines the number of electron-neutral collisions. The possible options are: 0, 1, 2, and 3. The default, 0, disables this feature. The list of electron-neutral collisions that can be modeled are:

1. elastic — Elastic interaction of electron particle with the background neutral gas. The incident electron gets scattered after the collision.
2. excitation — Inelastic interaction. The incident electron excites the neutral particle (i.e., the background gas) and loses the excitation threshold energy specified by the user and gets scattered after the collision.
3. ionization — Inelastic interaction. The incident electron ionizes the neutral particle (i.e., the background gas) and creates an ion and a secondary electron. The incident electron loses part of its energy (equivalent to the ionization threshold energy specified by the user) and gets scattered after the collision.

The users can include just one collision type or two collision types or all three collision types in the way they would prefer to model the electron-neutral collisions in their OOPIC Pro simulation.

**ElecCollXSectionFile (default = NULL)** This parameter specifies the file that contains the user's electron-neutral collision cross section data. The data file should be in plain text (ASCII) and contain the following:

- Atomic number of the neutral gas (integer)
- Collision name (string) — one of ELASTIC, EXCITATION or IONIZATION, describing the type of collision.
- Collision threshold energy (float) — a floating point value
- Number of collision data points (integer) — an integer value
- The collision cross section information for this collision in two columns of floating point values, separated by whitespace. The first column contains the incident electron energy values in eV units, and the second column contains the respective electron-neutral collision cross section values in  $m^2$  units.

The example file `electronCSection.dat` given below demonstrates how to set up the user's electron-neutral collision cross section data. The cross section values given here are for illustration and do not reflect any actual data.

```

18
ELASTIC
0.0
10
2.0 3.5e-19
5.0 5.e-19
10.0 6.e-19
15.0 5.5e-19
17.0 4e-19

```

```

20.0 3.e-19
50.0 2.e-19
100.0 1.e-20
250.0 1.e-21
1000.0 1.e-22
EXCITATION
12.0
10
14.0 3.5e-19
18.0 5.e-19
20.0 6.e-19
25.0 5.5e-19
32.0 4e-19
40.0 3.e-19
60.0 2.e-19
100.0 1.e-20
250.0 1.e-21
1000.0 1.e-22
IONIZATION
15.7
10
16.0 3.5e-19
18.0 5.e-19
20.0 6.e-19
25.0 5.5e-19
32.0 4e-19
40.0 3.e-19
60.0 2.e-19
100.0 1.e-20
250.0 1.e-21
1000.0 1.e-22

```

**NIonCollisions (default = 0)** A variable for defining the number of ion-neutral collisions. The possible options are: 0, 1, and 2. In order to use the user defined ion-neutral collision feature in the OOPIC Pro, the value of NElectronCollisions should be greater than or equal to 1. The list of ion-neutral collisions that can be modeled are:

1. ION\_ELASTIC — Elastic interaction. The incident ion gets scattered after the collision.
2. CHARGE\_EXCHANGE — Inelastic interaction. In this collision the incident ion exchanges the charge to the colliding neutral and turns into a neutral. Since the neutral gas is treated as a non moving background, the newly formed ion's energy after the collision event is determined based on the Maxwellian distribution function of the background neutral gas temperature.

The users can include either two collision types or just one collision type in the way they would prefer to model the ion-neutral collisions in their OOPIC Pro simulation.

**IonCollXSectionFile (default = NULL)** Name of the file that contains the user's ion-neutral collision cross section data.

The data file should contain the following:

- Atomic number of the neutral gas (integer)
- Collision name (string) — one of ION\_ELASTIC or CHARGE\_EXCHANGE
- Number of collision data points (integer)
- The collision cross section for this collision in two columns of floating point values, separated by whitespace. The first column contains the incident ion energy values in eV and the second column contains the respective ion-neutral collision cross section values in  $m^2$ .

The example file given below demonstrates how to set up user-defined ion-neutral collision cross section data. The cross sections data given are for illustration and does not reflect any actual data.

```

18
ION_ELASTIC
10
2.0    3.5e-19
5.0    5.e-19
10.0   6.e-19
15.0   5.5e-19
17.0   4e-19
20.0   3.e-19
50.0   2.e-19
100.0  1.e-20
250.0  1.e-21
1000.0 1.e-22
CHARGE_EXCHANGE
10
14.0   3.5e-19
18.0   5.e-19
20.0   6.e-19
25.0   5.5e-19
32.0   4e-19
40.0   3.e-19
60.0   2.e-19
100.0  1.e-20
250.0  1.e-21
1000.0 1.e-22

```

This feature of MCC blocks makes it possible to model any background gas type in OOPIC Pro. If the user wants to model a gas type that is not available in OOPIC Pro's default list of background gases (H, He, Ar, Ne, N, Xe, and Li) users can specify their own background gas cross sections. Note that if user-defined cross sections are used, the `gas` parameter in the MCC block should be set to H, instead of defining the gas type with their own gas name; this will avoid complaint error messages reported for the unknown gas type and leading to a crash of OOPIC Pro.

### 6.3.4 Load

These parameters specify the initial spatial and Maxwellian velocity distributions of each particle species that has been defined.



```
density = 1.0e14
np2c = 1e5
temperature = 5.93e5
}
```

For a complete example showing use of the `Load` block, see the input file `dcdis.inp`.

### 6.3.5 `VarWeightLoad`

`VarWeightLoad` provides the same parameters as `Load` but adjusts their values to keep the macroparticle density uniform in cylindrical space by linearly increasing the weighting factor `np2c` with the radius. An example of using *VarWeightLoad* can be found in `beamplasmatest.inp`.

### 6.3.6 `PlasmaSource`

A plasma source is a rectangular region in which plasma is generated at a given rate. The plasma in this case can consist of two species of particles that have been defined in a `Species` element. Each species in the plasma can be assigned its own properties regarding velocity, temperature, etc. Use multiple `PlasmaSource` elements if more than two species are needed. If the `analyticF` is the same then all species will originate within the same time and volume.

Table 10: PlasmaSource Parameters

Parameters	Default value	Data type	Units	Description
speciesName1	NULL	string		A species defined in a Species element
units1	MKS	string	MKS or eV	Units for ALL the velocities for species 1
v1drift1	0.0	scalar	m/s or eV	Drift velocity for x1.
v2drift1	0.0	scalar	m/s or eV	Drift velocity for x2.
v3drift1	0.0	scalar	m/s or eV	Drift velocity for x3.
temperature1	0.0	scalar	m/s or eV	Isotropic temperature (see Load)
v1thermal1	0.0	scalar	m/s or eV	Thermal velocity in x1 (see Load)
v2thermal1	0.0	scalar	m/s or eV	Thermal velocity in x2 (see Load)
v3thermal1	0.0	scalar	m/s or eV	Thermal velocity in x3 (see Load)
Ucutoff1	0.0	scalar	m/s or eV	isotropic upper cutoff for temperature.
v1Ucutoff1	0.0	scalar	m/s or eV	upper cutoff velocity in x1.
v2Ucutoff1	0.0	scalar	m/s or eV	upper cutoff velocity in x2.
v3Ucutoff1	0.0	scalar	m/s or eV	upper cutoff velocity in x3.
Lcutoff1	0.0	scalar	m/s or eV	isotropic lower cutoff for temperature.
v1Lcutoff1	0.0	scalar	m/s or eV	lower cutoff velocity in x1.
v2Lcutoff1	0.0	scalar	m/s or eV	lower cutoff velocity in x2.
v3Lcutoff1	0.0	scalar	m/s or eV	lower cutoff velocity in x3.
speciesName2	NULL	string		A species defined in a Species element
units2	MKS	string	MKS or eV	Units for ALL the velocities for species 1
v1drift2	0.0	scalar	m/s or eV	Drift velocity for x1.
v2drift2	0.0	scalar	m/s or eV	Drift velocity for x2.
v3drift2	0.0	scalar	m/s or eV	Drift velocity for x3.
temperature2	0.0	scalar	m/s or eV	Isotropic temperature (see Load)
v1thermal2	0.0	scalar	m/s or eV	Thermal velocity in x1 (see Load)
v2thermal2	0.0	scalar	m/s or eV	Thermal velocity in x2 (see Load)
v3thermal2	0.0	scalar	m/s or eV	Thermal velocity in x3 (see Load)
Ucutoff2	0.0	scalar	m/s or eV	isotropic upper cutoff for temperature.
v1Ucutoff2	0.0	scalar	m/s or eV	upper cutoff velocity in x1.
v2Ucutoff2	0.0	scalar	m/s or eV	upper cutoff velocity in x2.
v3Ucutoff2	0.0	scalar	m/s or eV	upper cutoff velocity in x3.
Lcutoff2	0.0	scalar	m/s or eV	isotropic lower cutoff for temperature.
v1Lcutoff2	0.0	scalar	m/s or eV	lower cutoff velocity in x1.
v2Lcutoff2	0.0	scalar	m/s or eV	lower cutoff velocity in x2.
v3Lcutoff2	0.0	scalar	m/s or eV	lower cutoff velocity in x3.
sourceRate	0.0	scalar	#/m3/s	rate of plasma generation m-3 s-1
np2c	1.e6	scalar		ratio of physical to computer particles
analyticF	"0.0"	string		Spatial distribution of the generated particles. Understands the variables x1, x2, and t.

## 6.4 Boundary Conditions

This collection of elements included within the Region block supports the definition of a number of physical boundary configurations that can be encountered in plasma modeling. Common to all possible configurations, there is a generic set of parameters that is used to specify geometric and time-dependent properties of the boundaries.

### 6.4.1 Generic Boundary Conditions (Boundary Geometry)

These parameters are used to define the endpoints of a line. The orientation of the material bounded by this line segment is determined by the value of the normal parameter (see below).

Table 11: Grid Spacing Parameters

Parameters	Default value	Data type	Units	Description
j1	-1	Int	Grid spacing	x1 index for first boundary endpoint.
k1	-1	Int	Grid spacing	x2 index for first boundary endpoint.
j2	-1	Int	Grid spacing	x1 index for second boundary endpoint.
k2	-1	Int	Grid spacing	x2 index for second boundary endpoint.

Alternatively, endpoints may be expressed in MKS units. However, OOPIC Pro will shift the computational boundary to coincide with the nearest grid point. The computational grid is specified in the `Grid` element.

Table 12: Grid Spacing MKS Parameters

Parameters	Default value	Data type	Units	Description
A1	-1	scalar	m	x1 location for first boundary endpoint.
A2	-1	scalar	m	x2 location for first boundary endpoint.
B1	-1	scalar	m	x1 location for second boundary endpoint.
B2	-1	scalar	m	x2 location for second boundary endpoint.

Boundary endpoints can also be specified by using the `Segment` specifier within the definition of a boundary type if the material all along the multi-segmented boundary is the same. This aids the definition of boundary condition geometries of a complex nature that require the definition of multiple, connected line segments to specify the material boundary. It is also an aid to efficient computation. Complex geometries are defined by multiple `Segment` specifiers. Not all boundary types can utilize multiple `Segment` specifiers. This situation is noted in the explanation of each boundary type that follows.

Input Block Example:

```
BoundaryGroup
{
  // generic and material parameters
  Segment
  {
    SegName = element1
    A1 = 0
    B1 = 0.3
    A2 = 0
    B2 = 0
    normal = 1
  }
}
```

```
    }  
  
    Segment  
    {  
        SegName = element2  
        A1 = 0.3  
        B1 = 0.3  
        A2 = 0  
        B2 = 0.3  
        normal = -1  
    }  
    // other generic and material parameters  
}
```

## 6.4.2 Generic Boundary Conditions (Boundary Material Properties)

Table 13: Species Parameters

Parameters	Default value	Data type	Units	Description
name	"Noname"	string		A name for this boundary.
normal	1	int		Unit direction of the face normal: -1=down /left, 1=up/right. For oblique segments the unit direction of the normal is: 1 for right face of boundary, -1 for left face of boundary
fill	0	int		If 1, the boundary is filled in. This implies a closed boundary.
EFFlag	0	flag		Data accumulator for Poynting flux through boundary: 0=off; 1=on.
IdiagFlag	0	flag		Data accumulator for particle current: 0=off; 1=on.
Ihist_avg	1	int		Number of time steps for averaging current plots.
Ihist_ien	1024	int		Length of the current history arrays.
nxbins	0	int		Number of spatial bins along boundary segment for distribution accumulation.
nenergybins	0	int		Number of energy bins for distribution accumulation.
nthetabins	0	int		Number of bins for angular distribution function.
theta_min	0	scalar		Minimum angle for angular distribution function.
theta_max	90	scalar		Maximum angle for angular distribution function.
energy_min	0	scalar	eV	Minimum energy for dist. function collector.
energy_max	100	scalar	eV	Minimum energy for dist. function collector.
C	1.0	scalar		DC value for time-dependent function.
A	0.0	scalar		AC value for time-dependent function.
frequency	0.0	scalar	Hz	Frequency of AC variation for time-dependent function.
phase	0.0	scalar	rad	Initial phase of AC variation for time-dependent function.
tdelay	0.0	scalar	s	Time to delay time before envelope function starts.
trise	0.0	scalar	s	Rise time of envelope for time-dependent function.
tpulse	10000.0	scalar	s	Pulse width of envelope for time-dependent function.
tfall	0.0	scalar	s	Fall time of envelope for time-dependent function.
a0	1.0	scalar		Base value of envelope for time dependent function.
a1	1.0	scalar		Maximum value of envelope for time-dependent function.
xtFlag	0	flag		Selects space or time dependence of boundary condition: 0 = envelope * sinusoid 1 = f(t) 2 = f(x) 3 = f1(x)*f2(t) 4 = f(x;t)
F	"100"	string		A string which is interpreted at runtime to give the space and/or time-dependence of this boundary
transparency	0	scalar		A coefficient for particle transmission between 0 and 1 that should allow you to set the transparency of the boundary. 0 would be fully opaque and 1 would be a fully transparent boundary.

### 6.4.3 Dielectric

To specify a boundary composed of a dielectric material all generic boundary parameters, as given above, are utilized as well as the following parameters for the specification of the material properties of the boundary. Only one `Segment` specifier within the `Dielectric` element is permitted for the definition of the physical boundary.

Table 14: Dielectric Parameters

Parameters	Default value	Data type	Units	Description
<code>er</code>	1.0	scalar		Relative permittivity
<code>QuseFlag</code>	1	Flag		Determines whether to use the accumulated surface charge in the field solve; i.e. 0 indicates that the charge drains off. This flag is only applicable to the electrostatic model.
<code>reflection</code>	0	scalar		Coefficient for particle reflection. Value is between 0 and 1.
<code>refMaxE</code>	1.e10	scalar	eV	Only reflect particles below this energy. <code>Secondary</code> and <code>Secondary2</code> groups can contain zero or more of these.

Input Block Example:

```
Dielectric
{
  er = 1.0
  j1 = 0
  j2 = 30
  k1 = 30
  k2 = 30
  name = bulb
  normal = -1
}
```

For a complete example, see the input file `xybeam.inp`.

### 6.4.4 Conductor

Specification of a boundary with conducting properties is accomplished by utilizing any or all of the generic parameters above in conjunction with those included in the `Dielectric` element. Multiple `Segment` specifiers are permitted.

Input Block Example:

```

Conductor
{
  j1 = 0
  k1 = 10
  j2 = 0
  k2 = 50
  normal = 1
}

```

A complete example using the `Conductor` block can be seen in the OOPIC Pro input file `klystron.inp`.

## 6.5 Temperature rise at the boundaries

A new feature has been included in the boundary diagnostic to measure the surface heating due to the impact of charge particles. This diagnostic uses the sum of the collected charge particle energy values at the boundary at each time step interval ( $\Delta t$ ) to measure the boundary surface's temperature rise  $\Delta T$ . It is given by

$$\Delta T = \frac{e_s \sum_{i=1}^n E_i}{A_s R C_p}$$

where  $e_s$  is the scale factor,  $E_i$  is the energy of the incident particle in Joules,  $n$  is the number of charge particles collected during  $\Delta t$ ,  $A_s$  is the boundary's surface area in  $m^2$ ,  $R$  is the material range in  $Kg/m^2$ , and  $C_p$  is the material specific heat in  $J/kg - K$ .

For the temperature rise diagnostic, following new parameters can be supplied by the user in their input file:

Table 15: Temperature Rise Diagnostic Parameters

Parameters	Default value	Data type	Description
<code>heatFluxParticlesFlag</code>	0	flag	Flag to turn on the temperature rise calculations.
<code>energyScaleFactor</code>	0.0	scalar	A scaling factor for determining the percentage of incident particle's energy contributions towards the temperature rise of the boundary surface.
<code>specificHeat</code>	385 J/kg-K, Copper material	scalar	Material specific heat value
<code>materialRange</code>	5.62e-5 $Kg/m^2$ , Copper	scalar	Material range value
<code>materialDensity</code>	8920 $Kg/m^3$ , Copper	scalar	Material density value

The following boundary block illustrates how the temperature rise diagnostic can be defined in an input file,

```

Conductor
{
  name = boundary1
  j1 = Nx
  k1 = 0
  j2 = Nx
}

```

```

k2 = Ny
normal = -1
nxbins = Ny
heatFluxParticlesFlag = 1
energyScaleFactor = 0.5 // 50% of the energy contributes to temperature rise
specificHeat = 385.0 // in J/(kg-K)
materialRange = 5.62e-5 // in Kg/(m^2)
materialDensity = 8920 // in Kg/(m^3)
}

```

### 6.5.1 Equipotential

Specification of an Equipotential boundary incorporates the same set of parameters defined for a `Conductor` boundary segment.

Note that an Equipotential boundary will use the time-dependent function parameters in the generic boundary parameter set to define a spatially uniform time-dependent voltage on a surface. This functions as a grounded `Conductor` in the electromagnetic model.

If multiple boundary segments, all at the same potential, are part of the same equipotential surface then it is important for performance that they are all specified with same Equipotential element. However, the user should be cautious when defining multiple segments in a single boundary block. This setup may cause problems such as particles being crossing into the metal surfaces when they are interacting with the metal boundaries that are defined using multiple segments option. This problem will not occur if the user just uses only one segment in each Equipotential boundary block definition. If the user observes problems with the simulation when using multiple segments in a single boundary block, then the best option is to switch to using multiple boundary blocks where each boundary block is defined with only one segment.

The example below includes two `Segment` specifiers within the same Equipotential element. For computational efficiency, this is preferable to defining solitary `Segment` specifiers within two Equipotential elements.

Input Block Example:

```

Equipotential
{
  xtflag = 0 // using envelope * sinusoid
  C = -2000 // DC offset
  A = 500 // Amplitude of AC component
  frequency = 1E8 // AC frequency
  phase = 90 // AC phase
  trise = 0 // envelope rise time
  tpulse = 15e-6 // duration of nonzero envelope
  tfall = 1e-9 // fall of envelope
  a1 = 1 // max scale of envelope
  a0 = 0 // min scale of envelope
  Secondary // secondary electrons emitted upon impact of argon ions
  // with 30% probability
  {
    secondary = 0.3
    secSpecies = electrons
    iSpecies = argon
  }
}

```

```

Segment
{
  // left boundary of equipotential surface
  j1 = 0
  k1 = 0
  j2 = 0
  k2 = Kmax
  normal = 1
}
Segment
{
  // top boundary of equipotential surface
  j1 = 0
  k1 = Kmax
  j2 = jRightSideWall
  k2 = Kmax
  normal = -1
}
}

```

For a complete example that can be run in OOPIC Pro, see the input file `dcdis.inp`.

### 6.5.2 Polarizer

Specification of a Polarizer boundary incorporates the same set of parameters defined for a Conductor boundary segment.

Table 16: Polarizer Parameters

Parameters	Default value	Data type	Units	Description
transmissivity	0	scalar		Fraction (between 0 and 1) of particles which pass through the polarizer
Phi	90	String	degrees	Polarization angle with respect to the simulation plane. This parameter can be a function of distance along the polarizer boundary as well. $X = 0$ is defined as the lower- or left-most point of the polarizer, and $X = 1$ is the other end.

Input Block Example:

```

Polarizer
{
  name = Polarizer
  A1 = 0
  A2 = Rdrift
  B1 = Zmax
  B2 = Rdrift
}

```

```

    normal = -1
    Phi = 45 * pulse(X, Ramp1Start, Ramp1Length, HelixLength, Ramp2Length)
}

```

### 6.5.3 Secondary

A `Secondary` element can be nested within any boundary element specification to include the production of a secondary species. Production of the secondary species modeled in OOPIC Pro employs a phenomenological model of the quantum processes that occur in the surface of materials.

For an example of `Secondary` usage, see `Equipotential` above. The `Secondary` element implements simple step function with a turn-on threshold to model secondary emission.

Table 17: Secondary Parameters

Parameters	Default value	Data type	Units	Description
<code>secondary</code>	0	scalar		Secondary emission coefficient; can be > 1
<code>threshold</code>	0.5	scalar	eV	Emission threshold incident energy must exceed for emission to occur
<code>Eemit</code>	2	scalar	eV	Maximum energy of emitted secondary particles (uniformly distributed).
<code>secSpecies</code>	NULL	string		Name of the species to emit as secondary particles; must correspond to an existing species if <code>secondary</code> > 0.
<code>iSpecies</code>	NULL	string		Incident species to generate as secondary particles

### 6.5.4 Secondary2

The `Secondary2` element implements a Vaughan-based secondary production model that includes energy and angular dependence as well as a full emission spectrum including reflected and scattered primaries. This is a more accurate model of secondary particle production.

The coefficient of secondary emission is the ratio of the numbers of ejected to incident electrons. It is given by:

$$\delta(\epsilon, \theta) = \delta_{max0} \left(1 + k_{s\delta} \frac{\theta^2}{2\pi}\right) (w \exp(1-w))^k$$

where  $\epsilon$  is the incident energy of a particle and  $\theta$  is its angle of incidence measured with respect to the surface normal.  $k_{s\delta}$  is a surface smoothness parameter (described below),  $\delta_{max0}$  is the peak secondary emission coefficient corresponding to the energy  $\epsilon_{max0}$  and normal incidence. The exponent  $k$  is

$$k = \begin{cases} 0.62, & w < 1 \\ 0.25, & w \geq 1 \end{cases}$$

Values for  $k$  are derived from a curve-fit analysis.

$w$  is the normalized energy given by:

$$w = \frac{\epsilon - \epsilon_0}{\epsilon_{max0}(1 + k_{sw}\theta^2/2\pi) - \epsilon_0}$$

where  $\epsilon_0$  is the secondary emission threshold (as defined for the `Secondary` element above).  $k_{sw}$  is a surface-smoothness parameter similar to  $k_{s\delta}$  (both can vary between 0 for rough surfaces and 2 for polished surfaces).

The nature of particle interaction is determined by comparing a uniform random variable,  $R$ , whose values are between 0 and 1 to (`fReflected` + `fScattered`). A true secondary particle is generated from a Maxwellian distribution if (`fScattered` + `fReflected`) <  $R$ . Otherwise a single component of the incident particle's velocity has its sign reversed. Orientation of the reflecting surface determines which component is changed. Further, if  $R > \text{fReflected}$ , all components of the particle's velocity are scaled by uniform (0,1) random values.

Particle interaction obeys the rule `fScattered` + `fReflected` + `fSecondary2` = 1.

`Secondary2` utilizes the parameters defined for `Secondary`. Note that the maximum secondary coefficient for `Secondary2` is defined by `Secondary`.

Table 18: `Secondary2` Parameters

Parameters	Default value	Data type	Units	Description
<code>fReflectedi</code>	0	scalar		Fraction of emitted particles that are reflected primaries
<code>fScattered</code>	0	scalar		Fraction of emitted particles that are scattered primaries
<code>energy_max0</code>	0	scalar		eV Impact energy at maximum particle yield ( <code>e[max0]</code> )
<code>ks</code>	1	scalar		Surface roughness; $0 \leq ks \leq 2$ (0 is rough, 2 is smooth)

### 6.5.5 `Secondary3`

The `Secondary3` element implements sophisticated secondary emission models through the numerical routines available from the TxPhysics library. The `Secondary3` element allows modeling of both electron-induced and ion-induced type secondary emission problems.

The physical algorithms adopted for modeling the electron-induced secondary electron emission is based on the methodologies given by Furman and Pivi (see M. A. Furman and M. T. F. Pivi, Phys. Rev. ST Accel. Beams 5, 124404 (2002)). `Secondary3`'s electron-induced secondary emission model determines the number of secondaries to be emitted from the wall and the emitted secondary electrons initial energies based on the incident electron's energy and their incident angle relative to the normal of the wall surface. The ion-induced secondary emission model is based on the numerical methodologies given by Rothard, H. et al, Phys. Rev. A 41, 2521 (1990)). The `Secondary3` ion-induced secondary emission model allows two types of secondary emissions. 1) the user can adopt either a constant probability model (by defining `emissionProb` which must be greater than 0.0 and  $\leq 1.0$ ) or 2) a normal model (by defining `emissionProb` value is equal to 0.0). In a constant probability model a single secondary electron is emitted with probability value of "`emissionProb`" and independent of incident ion parameters (such as energy, incident angle, etc.). But in the normal model the number of secondaries is determined based on the incident ion energy, incident angle, material type, and based on the  $dE/dx$  models.

The `Secondary3` element considers all of the input parameters considered for the `Secondary` element. In addition it has the following new parameters:

Table 19: Secondary3 Parameters

Parameters	Default value	Data type	Units	Description
materialType	1	flag		A flag to define the material type of the incident wall boundary. Currently allows two material types 1 for Cu and 2 for Steel.
emissionProb	0.0	scalar		A probability parameter needed for the ion-induced secondary emission model. If emissionProb is set between 0 and 1, then a constant probability model is used and if it is set to zero then normal ion-induced secondary emission model is adopted.

An example below describes how the Secondary3 element (considers both electron-induced and ion-induced secondary emissions) is being adopted in an Equipotential boundary.

```
Equipotential
{
  name = WallSurface
  C = 100
  j1 = 0
  k1 = 0
  j2 = 0
  k2 = Ny
  normal = 1
  nxbins = Ny

  Secondary3
  {
    // electron-induced secondary emission
    secSpecies = SEElectrons
    iSpecies = electrons
    materialType = 1
  }

  Secondary3
  {
    // ion-induced secondary emission
    secSpecies = SEElectrons
    iSpecies = ions
    materialType = 1 // material type
    emissionProb = 0.5
  }
}
```

The example files secondary3\_IncElectrons.inp and secondary3\_IncIons.inp contain complete simulations using the Secondary3 block.

### 6.5.6 Sputter

The Sputter element allows modeling of ion-induced sputtering of wall materials using the numerical routines available in TxPhysics library. The Sputter element determines the sputtering yield (i.e., the number of sputtered

atoms) using the Yamamura model which depends on the nuclear component of stopping, mass ratio and surface binding energy. The energy distribution of the sputtered atoms is based on model given in "Modeling ion-induced electrons in the High Current Experiment, " by P. H. Stoltz, J. P. Verboncoeur, R. H. Cohen, A. W. Molvik, J.-L. Vay, and S. A. Veitzer, Phys. Plasmas 13, 056702 (2006).

Table 20: Sputter Parameters

Parameters	Default value	Data type	Units	Description
targetNumber	1	flag		The wall material atom type. Only the copper material type is supported at present.
targetTemperature	0.025	scalar	eV	emperature of the wall surface given in eV.

A sample Sputter element is given below. The Sputter element can be adopted in any boundary blocks in the input file.

```
Sputter
{
  secSpecies = CuAtoms      // sputtered atoms
  iSpecies = ions           // incident ion species
  targetNumber = 29         // material atom number
  targetTemperature = 0.03 // in eV
}
```

The input file `sputter_IncIons.inp` also contains an example of using the Sputter block.

### 6.5.7 DielectricRegion

Specification of a DielectricRegion boundary incorporates the same set of parameters defined for a Dielectric boundary segment. Only one Segment specifier is permitted.

For DielectricRegion, the points formed by (j1, k1) and (j2, k2) indicate opposite corners of a rectangular region separate from the model boundaries rather than a line segment as would be used for a boundary. This provides the means to define a body with dielectric properties completely contained within the modeled system but detached from its boundaries.

Input Block Example:

```
DielectricRegion
{
  er = 100.0
  j1 = 3 * Jmax/13
  k1 = 3 * Kmax/13
  j2 = 10 * Jmax/13
  k2 = 10 * Kmax/13
  QuseFlag = 1
}
```

### 6.5.8 DielectricTriangle

Specification of a `DielectricTriangle` boundary incorporates the same set of parameters defined for a `Dielectric` boundary segment. Only one `Segment` specifier is permitted.

Similar to `DielectricRegion`, this element allows a body to be defined within the model boundaries but with a triangular, rather than rectangular, shape. For `DielectricTriangle`, the points formed by  $(j1, k1)$  and  $(j2, k2)$  indicate the two vertices of the base of a right triangle. The third vertex is determined by the parameter `normal`. If `normal > 0`, the third vertex of the triangle is above the line. Otherwise, it will be below the line.

### 6.5.9 CurrentRegion

Specification of a `CurrentRegion` boundary incorporates the generic set of parameters plus the parameters in the table. Only one `Segment` specifier is permitted.

Table 21: `CurrentRegion` Parameters

Parameters	Default value	Data type	Units	Description
<code>Current</code>	1	scalar	Amp	Magnitude of the TOTAL current in the region.
<code>direction</code>	3	int		Direction of current: x1, x2 or x3
<code>currentFile</code>	NULL	string		File containing current distribution
<code>analyticF</code>	0.0	string		Analytic function of x1, x2 describing the current distribution

Input Block Example:

```

CurrentRegion
{
  name = InputSig
  analyticF = 1
  A1 = L * 0.5 + dZ
  A2 = Ro - 2 * dR
  B1 = L * 0.5 -dZ + GapLength
  B2 = Ro - 3 * dR
  C = 0
  A = 1000
  direction = 1
  frequency = 2 * PI * Freq
  a0 = 0
  a1 = 1
  tdelay = 0
  trise = 0
  tpulse = 100000 * DT tfall = 0
}

```

### 6.5.10 Iloop

The Iloop adds one additional scalar parameter, Current, which defines the magnitude of the current in loop in Amps. The default is 1.

Input Block Example:

```
Iloop
{
  j1 = 0
  j2 = 20
  k1 = 0
  k2 = 10
  Current = 1
  frequency = 1.5e9
  C = 0
  A = 1
  tpulse = 3.33e-10
  a0 = 0
  trise = 0
  tfall = 0
}
```

### 6.5.11 CylindricalAxis

Specification of a CylindricalAxis boundary incorporates the generic set of parameters. This element is necessary only in cylindrical coordinates if  $r=0$  is included in the region.

Input Block Example:

```
CylindricalAxis
{
  j1 = 0
  k1 = 0 // k1 and k2 have to be zero, since boundary is on the x1 axis
  j2 = Jmax
  k2 = 0
  normal = 1
}
```

## 6.6 Ports

This collection of elements included within the Region block supports the definition of a number of free-space (void of physical material) boundary configurations that can be encountered in plasma modeling. As with the specification of physical boundaries there is an inheritance of, or inclusion of, the generic boundary parameters outlined earlier in Section 6.4 of this guide.

### 6.6.1 ExitPort

An `ExitPort` is used when an EM wave or a particle is to be treated as if it were escaping the defined boundaries of the simulation.

Specification of a `ExitPort` boundary incorporates the generic set of Boundary Condition parameters plus the parameters in the table. Only one `Segment` specifier is permitted.

Table 22: `ExitPort` Parameters

Parameters	Default value	Data type	Units	Description
R	376.99	scalar	ohms	Resistive component of surface impedance.
L	0	scalar	H	Inductive component of surface impedance.
Cap	0	scalar	F	Capacitive component of surface impedance.
Rin	376.99	scalar	ohms	Resistive component of surface impedance for incoming wave.
Lin	0	scalar	H	Inductive component of surface impedance for incoming wave.
Capin	0	scalar	F	Capacitive component of surface impedance for incoming wave.

Input Block Example:

```
ExitPort
{
  j1 = 0
  k2 = 0
  j2 = 0
  k1 = 0.04
  normal = 1
  EFFlag = 1
  name = below beam
  C = 0
  A = 0
  frequency = 1
}
```

### 6.6.2 Gap

Specification of a `Gap` boundary incorporates the generic set of parameters. Only one `Segment` specifier is permitted. Note that a `Gap` boundary will use the time-dependent function parameters in the generic boundary parameter set to define a spatially uniform time-dependent voltage on a surface. This works to set a time-dependent field (V/m) uniformly in space across the gap.

Input Block Example:

```
Gap
{
  j1 = cav1gL/DeltaZ
```

```

k2 = Rmax/DeltaR
j2 = cavlgStumpL/DeltaZ
k1 = Rmax/DeltaR
normal = -1
A = -3.2e6
frequency = 1.34e9
phase = 0
EFFlag = 0
name = Gap Left
}

```

### 6.6.3 PortGauss

PortGauss launches two linearly polarized electromagnetic pulses in 2-D Cartesian geometry. The pulses are launched from a given boundary by controlling the temporal and spatial dependence of the electric field at that boundary. The transverse spatial profile is Gaussian along the boundary. The temporal variation can be either trapezoidal, Gaussian or a half-sine. Pulse 0 is linearly polarized in the y direction, while pulse 1 is linearly polarized in the z direction. The pulses are launched in accordance with paraxial theory, such that the waist (focus) is a given distance from the port.

The parameters of pulse 0 incorporate the generic set of parameters. Only one Segment specifier is permitted.

Table 23: PortGauss Parameters

Parameters	Default value	Data type	Units	Description
waveLeng_p0		scalar	m	wavelength of the EM wave
amp_p0		scalar	V/m	peak wave amplitude of electric field
spotSize_p0		scalar	m	The beam half width at the waist.
focus_p0		scalar	m	Displacement of the focus.
chirp_p0		scalar		
pulShp_p0		int		pulse shape: 0=trapezoid 1=Gaussian 2=half-sine 3=wide Gaussian
pulLeng_p0		scalar	m	Pulse length: half-width for Gaussian, full for half-sine and trapezoid
tdelay_p0		scalar	m	time delay to start pulse.
offset	0	scalar	m	offset from the center of the simulation to the peak of the Gaussian

The parameters of pulse 1 (linearly polarized in the z direction) are the same, except that "\_p0" is replaced by "\_p1". To have only one pulse, set the amplitude of the other to zero.

OOPIC Pro will abort if the value of focus is set to zero. Set the value of the focus equal to one or two cell lengths inside the region (Dx or 2\*Dx) if the focus needs to be at the boundary.

For pulShp\_p0 = 1 (Gaussian shape), at  $t = \text{tdelay} \pm \text{tpulse}/2$ , the value is  $\exp(-1)$ , and zero beyond that. For pulShp\_p0 = 3 (wide Gaussian shape), the pulse is actually cut off further out (meaning the pulse is 6x longer), so the minimum value is  $4.8e-6$  with respect to the maximum.

Input Block Example:

```

PortGauss
{
  j1 = 0
  k1 = 0
  j2 = 0
  k2 = Ny
  normal = 1
  A = 0
  C = 1.0
  // Wave (0)
  pulShpp0 = nPulseShape
  tdelayp0 = 0.0
  pulLengp0 = pulseLength
  chirpp0 = 0.0
  spotSizep0 = waistSize
  waveLengp0 = laserWavelength
  focusp0 = waistLocation
  ampp0 = 0.0
  // Wave (1)
  pulShpp1 = nPulseShape
  tdelayp1 = 0.0
  pulLengp1 = pulseLength
  chirpp1 = 0.0
  spotSizep1 = waistSize
  waveLengp1 = laserWavelength
  focusp1 = waistLocation
  ampp1 = peakElectricField
  EFFlag = 0
  name = PortGauss
}

```

## 6.7 Particle Emitters

This set of elements are used to specify a variety of ways in which particles may be created in the modeled system.

### 6.7.1 Generic Emitter Parameters

Specification of generic parameters for particle emitters incorporates the specification of Dielectric Boundary Condition parameters. Only one `Segment` specifier is permitted.

This set of parameters is very similar to that used for `Load`.

Table 24: Generic Emitter Parameters

Parameters	Default value	Data type	Units	Description
units	MKS	string	MKS or EV	units for ALL the velocities
v1drift	0	scalar	m/s or eV	Drift velocity in x1
v2drift	0	scalar	m/s or eV	Drift velocity in x2
v3drift	0	scalar	m/s or eV	Drift velocity in x3
temperature	0	scalar	m/s or eV	Isotropic temperature
v1thermal	0	scalar	m/s or eV	Thermal velocity in x1
v2thermal	0	scalar	m/s or eV	Thermal velocity in x2
v3thermal	0	scalar	m/s or eV	Thermal velocity in x3
Lcutoff	0	scalar	m/s or eV	Isotropic lower cutoff for temperature
v1Lcutoff	0	scalar	m/s or eV	Lower cutoff velocity in x1
v2Lcutoff	0	scalar	m/s or eV	Lower cutoff velocity in x2
v3Lcutoff	0	scalar	m/s or eV	Lower cutoff velocity in x3
Ucutoff	0	scalar	m/s or eV	Isotropic upper cutoff for temperature
v1Ucutoff	0	scalar	m/s or eV	Upper cutoff velocity in x1
v2Ucutoff	0	scalar	m/s or eV	Upper cutoff velocity in x2
v3Ucutoff	0	scalar	m/s or eV	Upper cutoff velocity in x3
np2c	1.0e6	scalar		Number of physical particles to computer particles
speciesName	NULL	string		Refers to the Species with the same speciesName

### 6.7.2 BeamEmitter

Specification of `BeamEmitter` boundary incorporates the generic Emitter set of parameters. Only one `Segment` specifier is permitted.

Table 25: BeamEmitter Parameters

Parameters	Default value	Data type	Units	Description
I	1	scalar	A	Emission current, Note: In the case of Cartesian coordinates, I refers to A/m since the ignored (z-direction) dimension length is taken as 1 m.
thetadot	0	scalar	rad/s	Angular velocity of beam
nIntervals	0	int		Number of intervals for inversion of spatial dependence of $J(x)$ as specified by F. Zero means use the number of cells.

Input Block Example:

```
BeamEmitter
{
  j1 = 0
  k1 = 0
  j2 = 0
  k2 = 10*Kmax/13
```

```

normal = 1
speciesName = electrons
thetadot = 0.0
I = 10
np2c = 1.0E7
v1drift = 1e8
v2drift = 0
v3drift = 0
v1thermal = 0
v2thermal = 0
v3thermal = 0
}

```

### 6.7.3 EmitPort

EmitPort is a subclass of BeamEmitter. It applies dielectric boundary conditions to the port segment rather than conductor boundary conditions.

Input Block Example:

```

EmitPort
{
  j1 = 9
  j2 = 11
  k1 = 20
  k2 = 20
  speciesName = electrons
  normal = -1
  np2c = 4e3 // number of electrons per macroparticle
  I = 0.000005 // current
  v2drift = 0.01
}

```

### 6.7.4 VarWeightBeamEmitter

Specification of VarWeightEmitter boundary incorporates the BeamEmitter (see Section 6.7.1) set of parameters. Only one Segment specifier is permitted. This segment is used in cylindrical coordinates. It increases the np2c weight linearly to maintain a uniform beam density.

Table 27: VarWeightBeamEmitter Parameters

Parameters	Default value	Data type	Units	Description
nEmit	0	int		Number of particles per time step to emit; if nEmit=0 then the number is computed using np2c.

### 6.7.5 FowlerNordheimEmitter

This block implements the Fowler-Nordheim law for field emission according to the following equation:

$$J_{FN} = \frac{A_{FN}(\beta_{FN}E)^2}{\Phi_w} \exp\left(-\frac{B_{FN} v(y)\Phi_w^{3/2}}{\beta_{FN}E}\right)$$

where  $J_{FN}$  is the emitted current density in  $A/m^2$ ,  $E$  is the perpendicular component of the electric field in  $V/m$ , and  $\Phi_w$  is the work function of the material in eV. The other parameters have the following value or functional form:

$$y = C_y E^{1/2} / \Phi_w$$

$$v(y) = 1 - C_v y^2$$

$$A_{FN} = 1.5414 \times 10^{-6}$$

$$B_{FN} = 6.8308 \times 10^9$$

An instance of this new field-emission surface is created via an input block with the name `FowlerNordheimEmitter`. A `FowlerNordheimEmitter` incorporates the generic `Emitter` set of parameters.

Table 28: `FowlerNordheimEmitter` Parameters

Parameters	Default value	Data type	Units	Description
AFN	1.5414e-6	scalar		Coefficient A of the Fowler-Nordheim field-emission model
betaFN	1.0	scalar		Coefficient beta of the Fowler-Nordheim field-emission model
BFN	6.8308e9	scalar		Coefficient B of the Fowler-Nordheim field-emission model
CvFN	0.0	scalar		Coefficient $C_v$ of the Fowler-Nordheim field-emission model
CyFN	3.79e-5	scalar		Coefficient $C_y$ of the Fowler-Nordheim field-emission model
Phi_wFN	4	scalar	eV	Coefficient Phi_w of the Fowler-Nordheim field-emission model
nIntervals	0	scalar		Number of intervals to be used for emitting particles. nIntervals will be reset to the number of cells along the emitting boundary (with a minimum of 2)

An example of using `FowlerNordheimEmitter` appears below.

```

FowlerNordheimEmitter
// Below, we specify all of the Fowler-Nordheim parameters that are
// specific to this type of particle emitter, even though most of
// them are given the default value.
{
  j1 = (numCellsX - numEmitterCells) / 2
  j2 = (numCellsX + numEmitterCells) / 2
  k1 = numCellsY
  k2 = numCellsY
  normal = -1
  speciesName = electrons // name from species group above
  np2c = 5.e+6 // numerical weight of emitted particles
  // Coefficient "A" of the Fowler-Nordheim field emission model.
  // The default value is 1.5414e-06, which is specified here.
  AFN = 1.5414e-06
  // Coefficient "beta" of the Fowler-Nordheim field emission model.
  // This simply multiplies the electric field from the simulation.
  // The default value is 1. Here, we specify beta_FN = 20000, which
  // yields non-zero field emission, without having large electric // fields.
  betaFN = 20000.
  // Coefficient "B" of the Fowler-Nordheim field emission model.
  // The default value is 6.8308e+09, which is specified here.
  BFN = 6.8308e+09
  // Coefficient "Cv" of the Fowler-Nordheim field emission model.
  // The default value is 0, which is specified here.
  CvFN = 0.
  // Coefficient "Cy" of the Fowler-Nordheim field emission model.
  // The default value is 3.79e-05, which is specified here.
  CyFN = 3.79e-05
  // The work function "Phiw" for electrons in the surface, in eV.
  // The default value is 4 eV, which is specified here.
  PhiwFN = 4.
  // The number of intervals to be used for emitting particles.
  // The default value of 0, which is specified here.
  // In the default case, nIntervals will be reset to the # of cells
  // along the emitting boundary (with a minimum of 2), which is
  // the most reasonable thing to do.
  nIntervals = 0
}

```

## 6.8 Diagnostics

### 6.8.1 Default Diagnostics

A large set of pre-determined diagnostics is available for display. Additional diagnostics may be requested according to specifications in the input file. The format for requesting additional diagnostics is given in Section 6.1.4.

The following diagnostics are defined by OOPIC Pro (in the order they appear in the diagnostics window). Refer to the definition of parameters in the `Control` specifier for further information on the parameters that control the inclusion of components in this list. The labels used for the designation of coordinate and diagnostic components are explained in Section 4.5.2. For example, let the designations 1, 2, and 3 relate to the coordinate axes of the coordinate system used to define the simulation.

- As a function of x1, x2 and x3:
  - E1, E2 and E3
  - B1, B2 and B3
  - I1, I2, and I3 if ElectroStaticFlag is not set
  - $U_e$  (electric field energy density) and  $U_b$  (magnetic field energy density), defined as  $U_e = \frac{1}{2}\epsilon_0 t^2$  and  $U_b = \frac{1}{2}(\frac{B^2}{\mu_0})$ , with units of  $J/m^3$  (or equivalently,  $N/m^2$ )
- As a function of x1 and x2:
  - Poynting Vector if ElectroStaticFlag is not set
- As a function of x1, x2 and x3:
  - Rho
  - Densities of any neutral gases present
  - Phi if ElectroStaticFlag is set
  - Divergence Error if ElectroStaticFlag is not set
- Position plots for all species
- Phase space plots for all species for all combinations of position vs. velocity:
  - U1 vs. x1
  - U2 vs. x1
  - U3 vs. x1
  - U1 vs. x2
  - U2 vs. x2
  - U3 vs. x2
- As a function of x1 and x2:
  - E
  - B
  - I, if ElectroStaticFlag is not set
- Time domain plots
  - Average magnitude of divergence error if ElectroStaticFlag is not set
  - Average kinetic energy vs. time
  - Number
  - Kinetic Energy
  - Total density
  - If IdiagFlag is set: I history for boundary, I local history for boundary
  - If EFFlag is set: Poynting history for boundary, Poynting local history for boundary
- Composite plot of Poynting history
  - Potential and kinetic energies
  - E and B field energies

- RMS beam parameters for species with rmsBeamSizeFlag set
  - Average beam size
  - RMS beam size
  - Average velocity
  - RMS velocity
  - RMS emittance
  - Average energy
  - RMS energy
- Number density for each species
- Initial number density for each species if Show\_Loaded\_DensityFlag set
- Boundary diagnostics as defined by user

### 6.8.2 Temperature rise at the boundaries

A new feature has been included in the boundary diagnostic to measure the surface heating due to the impact of charge particles. This diagnostic uses the sum of the collected charge particle energy values at the boundary at each time step interval ( $\Delta t$ ) to measure the boundary surface's temperature rise  $\Delta T$ . It is given by

$$\Delta T = \frac{e_s \sum_{i=1}^n E_i}{A_s R C_p}$$

where  $e_s$  is the scale factor,  $E_i$  is the energy of the incident particle in Joules,  $n$  is the number of charge particles collected during  $\Delta t$ ,  $A_s$  is the boundary's surface area in  $m^2$ ,  $R$  is the material range in  $Kg/m^2$ , and  $C_p$  is the material specific heat in  $J/kg - K$ .

For the temperature rise diagnostic, following new parameters can be supplied by the user in their input file:

Table 29: Temperature Rise Diagnostic Parameters

Parameters	Default value	Data type	Description
heatFluxParticlesFlag	0	flag	Flag to turn on the temperature rise calculations.
energyScaleFactor	0.0	scalar	A scaling factor for determining the percentage of incident particle's energy contributions towards the temperature rise of the boundary surface.
specificHeat	385 J/kg-K, Copper material	scalar	Material specific heat value
materialRange	5.62e-5 Kg per $m^2$ , Copper	scalar	Material range value
materialDensity	8920 Kg per $m^3$ , Copper	scalar	Material density value

The following boundary block illustrates how the temperature rise diagnostic can be defined in an input file,

```

Conductor
{
  name = boundary1
  j1 = Nx
  k1 = 0
  j2 = Nx
  k2 = Ny
  normal = -1
  nxbins = Ny
  heatFluxParticlesFlag = 1
  energyScaleFactor = 0.5 // 50% of the energy contributes to temperature rise
  specificHeat = 385.0 // in J/(kg-K)
  materialRange = 5.62e-5 // in Kg/(m^2)
  materialDensity = 8920 // in Kg/(m^3)
}

```

### 6.8.3 User-Defined Diagnostics

Along with the default diagnostics available during an OOPIC Pro simulation, new diagnostics can be created which can also be plotted when running OOPIC Pro.

A new diagnostic can be specified in an element with the heading ‘Diagnostic’. The value of the diagnostic is put into the parameter VarName. The point, line, or region over which the diagnostic is to be sampled must be specified. User-defined diagnostics will dump time-averaged data.

Table 30: Diagnostic Geometry Parameters

Parameters	Default value	Data type	Units	Description
j1	-1	int		x1 index for first diagnostic endpoint.
k1	0	int		x2 index for first diagnostic endpoint.
j2	0	int		x1 index for second diagnostic endpoint.
k2	k1	int		x2 index for second diagnostic endpoint.

As with the generic boundary conditions, diagnostic boundaries can be specified in MKS units. However, OOPIC Pro will put the diagnostic on the nearest grid point.

Table 31: Diagnostic Parameters

Parameters	Default value	Data type	Units	Description
A1	0	scalar	m	x1 location for first diagnostic endpoint.
A2	0	scalar	m	x2 location for first diagnostic endpoint.
B1	0	scalar	m	x1 location for second diagnostic endpoint.
B2	0	scalar	m	x2 location for second diagnostic endpoint.

The type of graph plotted by OOPIC Pro for the diagnostic will be determined by the diagnostic boundaries. There are three types of graphs:

**Spatial Regions** If  $(j_1, k_1)$  and  $(j_2, k_2)$  form the corners of a rectangle then the graph is X versus Y or R versus Z. This is good for plotting the time average of a variable.

**Time History of a Line** If  $(j_1, k_1)$  and  $(j_2, k_2)$  form the ends of a line, then the graph is a time history of the variable versus position along the line.

**Time History of a Point** If  $(j_1, k_1)$  and  $(j_2, k_2)$  are the same point then the graph is a time history of the variable at that point.

Other parameters:

Table 32: Parameters for Time History of a Point

Parameters	Default value	Data type	Units	Description
VarName	'NULL'	string		Name of the variable chosen from list (below) to be plotted.
x1_Label	x1	string		x1 Label of the plot
x2_Label	x2	string		x2 Label of the plot
x3_Label	x3	string		x3 Label of the plot
title	'not_named'	string		Title of window.
HistMax	64	int		maximum length of history array
Comb	0	int		Every Comb'th value is left when HistMax is reached.
Ave	0	int		Averaged over Ave data points when adding to history array
integral	NULL	string		one of: line (variable dotted into dl); flux (variable dotted into dS); sum (simple summation)
polarizationEB	"EyBz"	string		"EyBz" computes diagnostic for $(E_y - c*B_z)/2$ and $(E_y + c*B_z)/2$ . "EzBy" specifies the calculation of $(E_z - c*B_y)/2$ and $(E_z + c*B_y)/2$ instead.
psd1dFlag	0	int		"psd1dFlag = 1" calculates the 1d power spectral densities for the two linear combinations of E and B selected via the "polarizationEB". The 1d PSD are calculated along the x axis for each value of the y index.
windowName	'Blackman'	string		Window to apply to data before the FFT. The following windows are implemented: "Blackman", "Bartlett", "Hamming", "Hann" and "Welch". (UNIX only)
nfft	0	scalar		Number of points to be used in fft computation (UNIX only)

The time histories are of length `HistMax`. When the array is full and `Comb` is defined then the array is combed to every `Comb`'th value. If `Comb` is 0 then the array is a moving window. If `Ave` is set to a nonzero value, `Ave` data points are averaged together.

Currently the following diagnostic variables can be plotted (default specification). Particle positions vs. time are displayed without having to be requested.

For time histories:

Table 33: Time History Diagnostic Parameters

Diagnostic Name	Content
E1	Ez (RZ) or Ex (XY)
E2	Er (RZ) or Ey (XY)
E3	Ephi (RZ) or Ez (XY)
B1	Bz (RZ) or Bx (XY)
B2	Br (RZ) or By (XY)
B3	Bphi (RZ) or Bz (XY)
I1	Iz (RZ) or Ix (XY) (only with EM field solve)
I2	Ir (RZ) or Iy (XY) (only with EM field solve)
I3	Iphi(RZ) or Iz(XY) (only with EM field solve)
intEdl1	Ez (RZ) or Ex (XY)
intEdl2	Er (RZ) or Ey (XY)
intEdl3	Ephi (RZ) or Ez (XY)
poynthing1	Poynting Vector in x1 (only with EM field solve)
poynthing2	Poynting Vector in x2 (only with EM field solve)
poynthing3	Poynting Vector in x3 (only with EM field solve)
Rho	total charge density
speciesName	charge density of a given species
phi	potential (only with electrostatic field solve)
Q	surface charge on dielectrics
LaserSpotSize	$\text{Integral}(y*y* Ey*Ey) / \text{Integral}(Ey * Ey)$ over the line k1 to k2, assuming the laser spot is centered at $(k2-k1)/2$ . Display is the average over HISTMAX time steps of this measure.
WaveDirDiagnostic	Computes $(Ey - c*Bz)/2$ and $(Ey + c*Bz)/2$ over the mesh which distinguishes left and right moving waves in the system (polarized in y).

For spatial regions, if JdotE is the requested diagnostic then JdotE is plotted.

An example of an input file using diagnostics is beamplasmatest.inp, which is included in the example input files.

Input Block Example:

```

Diagnostic
{
  j1 = 0 //geometry of diagnostic, in this case it is a 2-D region
  j2 = Nx
  k1 = 0
  k2 = Ny
  VarName = WaveDirDiagnostic
  polarizationEB = EzBy
  psdldFlag = 1 // calculate the 1d power spectral density
  windowName = Hann
  title = Wave Energy
  x1Label = x
  x2Label = y
  x3Label = Wave Energy
}

```

### 6.8.4 H5Diagnostic

OOPIC Pro will selectively dump E and B diagnostic data in HDF5 format. To call for a diagnostic to be dumped in HDF5 format, specify an element with the heading 'H5Diagnostic'. The value of the diagnostic is put into the parameter VarName. The point, line, or region over which the diagnostic is to be sampled must be specified. Note that unlike other data dumps in OOPIC Pro, user-defined H5Diagnostic data is *not* time-averaged, but records E and B directly.

Table 35: H5Diagnostic Parameters

Parameters	Default value	Data type	Description
j1	-1	int	x1 index for first diagnostic endpoint
ki	0	int	x2 index for first diagnostic endpoint
j2	0	int	x1 index for second diagnostic endpoint
k2	k1	int	x2 index for second diagnostic endpoint

As with the generic boundary conditions, diagnostic boundaries can be specified in MKS units. However, OOPIC Pro will put the diagnostic on the nearest grid point.

The type of graph plotted by OOPIC Pro for the diagnostic will be determined by the diagnostic boundaries. There are three types of data collected:

Table 36: Diagnostic Boundary Parameters

Parameters	Default value	Data type	Units	Description
A1	0	scalar	m	x1 location for first diagnostic endpoint
A2	0	scalar	m	x2 location for first diagnostic endpoint
B1	0	scalar	m	x1 location for second diagnostic endpoint
B2	0	scalar	m	x2 location for second diagnostic endpoint

**Spatial regions** If (j1, k1) and (j2, k2) form the corners of a rectangle then the graph is X versus Y or R versus Z. This is good for plotting the time average of variable.

**Time history of a line** If (j1, k1) and (j2, k2) form the ends of a line, then the graph is a time history of the variable versus position along the line.

**Time history of a point** If (j1, k1) and (j2, k2) are the same point then the graph is a time history of the variable at that point.

Other parameters:

Table 37: Other Diagnostic Parameters for Time History

Parameters	Default value	Data type	Description
VarName	'NULL'	string	Name of the variable chosen from list (below) to be dumped.
dumpPeriod	0	int	0 => dump at same frequency as binary dump file; nonzero = number of time steps between hdf5 dumps
filename	"Diagnostics"	string	Name of diagnostic HDF5 dump file. This will be set to <inputfilebase>.h5 before first dump occurs

Currently the following diagnostic variables can be dumped if WaveDirDiagnostic and PSDFieldDiag are specified in the input file.

For time histories:

Table 38: H5Diagnostic - Time History Diagnostics

Diagnostic Name	Content
E1	Ez (RZ) or Ex (XY)
E2	Er (RZ) or Ey (XY)
E3	Ephi (RZ) or Ez (XY)
B1	Bz (RZ) or Bx (XY)
B2	Br (RZ) or By (XY)
B3	Bphi (RZ) or Bz (XY)
I1	Iz (RZ) or Ix (XY) (only with EM field solve)
I2	Ir (RZ) or Iy (XY) (only with EM field solve)
I3	Iphi(RZ) or Iz(XY) (only with EM field solve)
ESPotential	phi (only with ES field solve)
NGD	Neutral gas density
SpName_numberDensity	Particle number density data for species named with "SpName" under Species block

### 6.8.5 Limit

A Limit rule has the form:

*variable op value*

where variable is the name of a parameter, value is a numeric value and op is an operator from the list '==, >=, <=, >' and '<'. An example of a Limit rule that would restrict the simulation time step to be nonzero and positive would be  $dt > 0$ .

### 6.8.6 Relation

A Relation rule has the form:

*variable1 op variable2*

where *variable1* and *variable2* are the names of two parameters (within a given `ParameterGroup`) and *op* is an operator from the same list defined for the Limit rules. This type of rule constrains two parameters to have a specified relationship. A rule requiring that the simulation time step to satisfy certain convergence criteria would be  $dt \leq Courant$ .

### 6.8.7 Algebra

An Algebra rule has the form:

*variable1 oparith variable2 op value*

where *variable1* and *variable2* are the names of two parameters (within a `ParameterGroup`), *oparith* is binary arithmetic operator from the list '+, -, \*, /' and *op* is an operator from the list given for the Limit rule.

### 6.8.8 Storing Time History Diagnostics

Now the user can store the time history results from an OOPIC Pro simulation in hdf5 format without using the graphical user interface. To enable this feature the user activates the `StoreTimeHistoryFlag` in the control block of OOPIC input file (`StoreTimeHistoryFlag = 1`). This flag will be handy for those users who would like to run the OOPIC Pro simulation in batch mode or not running OOPIC Pro interactively under both in single-processor (serial) and in multiple processor (parallel) runs. In the case of a parallel OOPIC Pro simulation, all of the time history results will be stored only in processor 0's (i.e., the first processor) H5 diagnostic file which contains the results for the entire computational domain.

The following options are available for storing the time history now (both in serial and parallel OOPIC Pro versions) using the `H5Diagnostic` options:

- Number of computer particles vs time history (will store all species type) (set `VarName = ncomputer_particle` in `H5Diagnostic`)
- Number of physical particles vs time history (will store all species) (set `VarName = nphysical_particle` in `H5Diagnostic`)
- Species average kinetic energy (in eV) vs time history (will store all species) (set `VarName = avgKE_species` in `H5Diagnostic`)
- Continuum radiated power vs time history (set `VarName = cont_radiatedPower` in `H5Diagnostic`)
- Line radiated power vs time history (set `VarName = line_radiatedPower` in `H5Diagnostic`)
- Total radiated power vs time history (set `VarName = total_radiatedPower` in `H5Diagnostic`)

The following boundary time history diagnostic option is available only in serial versions of OOPIC Pro.

- Temperature rise vs time history (set `VarName = temperatureRise_boundaryName` in `H5Diagnostic`)

Two examples :

```
H5Diagnostic
{
  VarName = ncomputer_particle // computer particle history to dump
  dumpPeriod = nsteps          // number of steps to dump the data
  fileName = DiagTimehistory   // the h5 file name of the diagnostic
}
H5Diagnostic
{
  // WallSurface's boundary temperature rise history
  VarName = temperatureRise_WallSurface
  dumpPeriod = nsteps          // number of steps to dump the data
  fileName = DiagTimehistory   // the h5 file name of the diagnostic
}
```

## 7 Example Solutions

This section provides examples of how OOPIC Pro can be applied to the modeling of a variety of different physical devices and phenomena.

### 7.1 A Simple Klystron

This example illustrates the use of OOPIC Pro in modeling the behavior of a simple klystron amplification device.

In the example given here, a two-cavity klystron is modeled in cylindrical coordinates on a 74 by 36 computational grid. The physical dimensions are 0.372 m along the axis of symmetry with a radius of 0.109 m. Figure 11 shows one half of an axial slice of the klystron that occupies the full computational grid. The full klystron would be a body achieved by rotating this diagram about  $r = 0$ . The input RF signal has an amplitude of 30 and a frequency of 1.21 GHz.

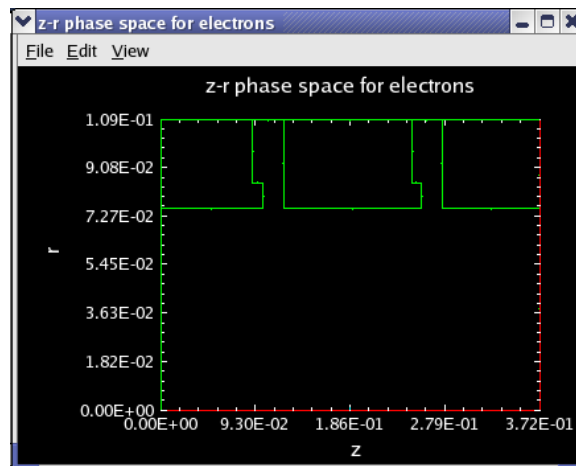


Figure 11: Klystron profile

The simulation parameters are defined in the file `klystron.inp`.

The graphs that follow are selected from the OOPIC Pro diagnostics list via the View → Diagnostics menu. They represent the state of the Klystron after 1500 time steps (about 7.5 nanoseconds). Figure 12 shows the position of the electron beam (shown in blue) as it emerges from the `BeamEmitter` component and impinges on the opposite wall of the klystron (traveling from left to right in this plot). Keep in mind that this diagram only shows one half of an axial slice so that to visualize the klystron in three dimensions this figure would have to be rotated about the axis of symmetry (along the bottom of the plot). This means that the electron beam itself is actually shaped like a hollow, thin-walled cylinder within the klystron. The bunching of electrons along the beam path is visibly evident in this plot near the input and output ports.

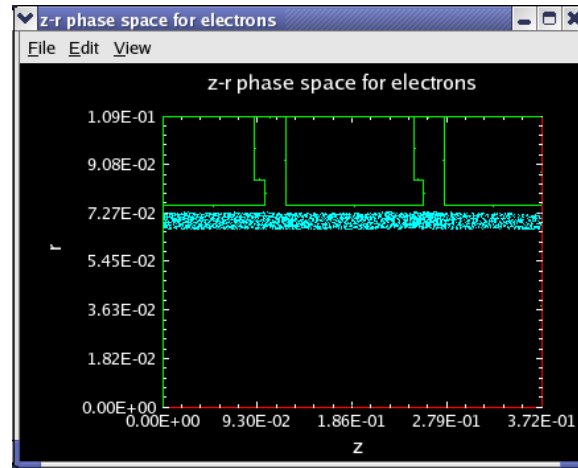


Figure 12: Cross-section of electron beam traversing the klystron

The influence of the modulating wave entering the klystron through the input cavity can be seen in the electron bunching (nonuniform density of blue points representing electron macroparticles) shown in Figure 12 and also in the phase plot shown in Figure 13. This scaled electron velocity plot displays the scaled axial velocity of electrons against their axial position. In this example the input drift velocity of the electrons is  $2.480 \times 10^8$  m/s. The scaled velocity is  $u = gv$  where

$$g = (1 - v^2/c^2)^{-1/2} = 1.780$$

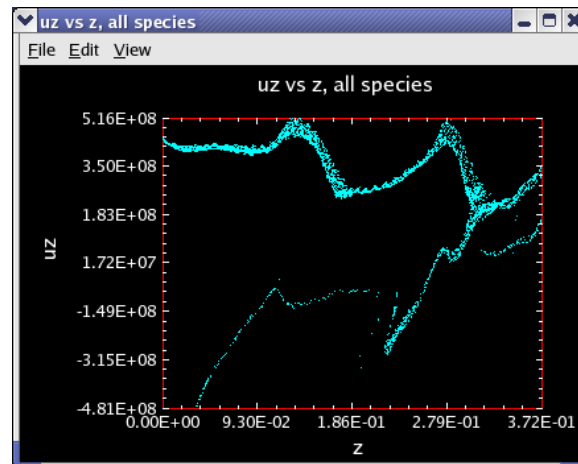


Figure 13: Scaled electron velocity as a function of axial displacement

By this point in time in the simulation the beam of electrons has been able to traverse both the input and the output port and impinge on the opposite klystron wall as well. Passing the input cavity subjects the electrons to the time-dependent forcing function which has been specified in the input file. The forcing function alternately slows and accelerates electrons that pass near the input cavity. This leads to the velocity profile seen in Figure 13 in which some electrons have a negative velocity along the axis and it can be seen that a general circulation (or apparent bunching) of electrons is occurring within the klystron.

Also of interest are history traces of the Poynting vector at the input and output cavity openings. Collection of this data is controlled by turning on the EFFlag parameter in the input file. At the Input Cavity/Klystron interface the variation in the Poynting vector is initially due to only the modulated input RF signal. Eventually electron circulation contributes as well.

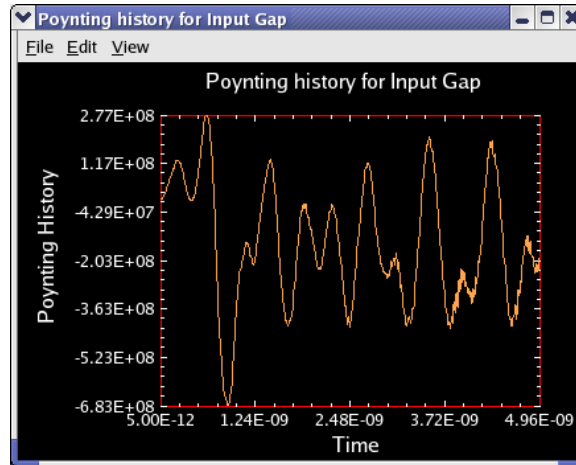


Figure 14: Poynting vector at the Input Cavity interface

At the Output Cavity, the Poynting vector magnitude (Figure 15) has a more regular periodic variation with an increasing magnitude. This illustrates the use of a klystron device as an RF amplifier.

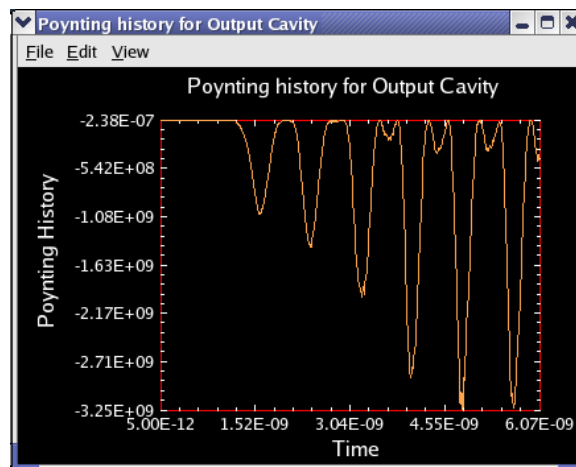


Figure 15: Poynting vector at the Output Cavity interface

## 7.2 Field-induced Tunneling Ionization by a Laser Pulse

This example (`loasisHe.inp`) illustrates the use of OOPIC Pro in modeling tunneling ionization due to propagation of a laser pulse into and through a neutral gas; Helium in this case. This example has been used to study phenomena such as the single and double ionization of Helium by laser impact and frequency up-shifting of the leading edge of the laser pulse. For more detail refer to "Simulations of Laser Propagation and Ionization in l'OASIS Experiments" and "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions."

The l'OASIS input file defines a 2-d simulation conducted in Cartesian coordinates. The x coordinate defines the longitudinal direction along which the laser pulse will propagate. The polarization of the pulse is in the z direction, the redundant coordinate. The pulse enters a "simulation box" from its left face and propagates to the right. A `PortGauss` element is defined at this boundary to launch the laser pulse. The input file further specifies that, in the transverse direction, the pulse will have a Gaussian shape. The two sides of the simulation box adjacent to the entry face are defined to be `ExitPorts`, and the opposite face is a `Conductor`. Due to symmetries, the simulation box can be represented by a single slice along the z plane that is overlaid with a computational two-dimensional grid 256 by 512 in size.

With the laser wavelength fixed, the sampling theorem requires that the laser wavelength span at least two grid points. Increasing this ratio improves the spatial resolution of the simulation but needs to be weighed against the computational resources available. A minimum time step can be computed from requirements to satisfy the Courant condition for adequate resolution of phenomena in the time domain once this set of geometric parameters has been established.

For this example, the laser wavelength  $\lambda = 0.8$  mm. With a grid size  $N[x] = 512$ ,  $N[y] = 256$  and 16 grids per wavelength for adequate spatial resolution, the time step is  $1.65 \times 10^{-16}$  sec. See the input file for details of this calculation. Most of the geometric parameters have been assigned to variables that are then used to calculate derived parameters such as the time step required to satisfy the Courant condition.

In order to be able to model a realistic experimental domain, OOPIC Pro provides a moving window capability that supports the adjustment of the numerical grid to center on physical phenomena of interest as the simulation progresses in time. In this example the geometry of the simulation is initially set up to reflect the simulation box that will be required to fully contain a single laser pulse for a finite time before it approaches the boundaries of a helium cloud (with sharp edges). As the laser penetrates the helium, the geometric boundaries of the simulation, as maintained by OOPIC Pro, shift along the direction of propagation in order to more accurately capture the physical phenomena occurring in and around the laser pulse. OOPIC Pro continues to move the simulation box along the direction of pulse propagation until the pulse has completely exited the helium cloud and the interaction is complete.

For this example, the density of helium changes from 0 to  $2 \times 10^{25} m^3$  over a distance of four grid cells, eight cells from the initial end of the simulation box. The moving window algorithm is activated when the pulse is within 15 cells of the end of the simulation box. The helium cloud extends a distance of two Rayleigh wavelengths along the propagation axis.

Figures 16 and 17 show the laser pulse in the first phase of the simulation. This reflects the time just prior to activation of the moving window algorithm when the laser pulse is about to impinge upon the helium cloud.

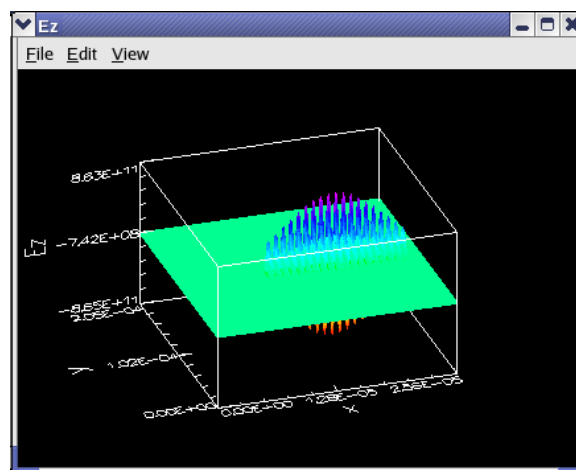


Figure 16: Laser pulse profile in vacuum

OOPIC Pro's moving window allows the definition of the simulation box to actually extend beyond the immediate

computational grid by following the laser pulse and ignoring the rest of the system for the computations that need to be performed to capture the current laser/gas interaction. This keeps computational load manageable and robust. The computational grid remains at the same manageable size but will cover sequential sections of the simulation box as it follows the propagation of the laser pulse. In the graphic output it will be seen that coordinates along the x direction do not change with time. This is because the coordinates shown relate to the moving window and not the full simulation box.

At time  $t = 0$  in the simulation the pulse is launched into a vacuum and propagates along the x direction into the simulation box. It eventually encounters the neutral He gas. The gas density profile is a linear ramp increasing quickly over a small number of grid cells along x from zero (vacuum) to the maximum density specified in the input file. The gas density ramps downward in a similar fashion after a distance along x. This allows the pulse to travel for some distance while totally immersed in the neutral gas. The pulse then exits the simulation through a vacuum at the far end of the simulation box.

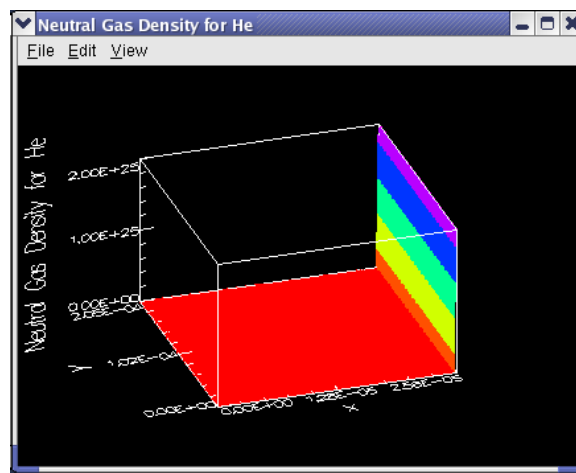


Figure 17: Profile of Helium density in the area of vacuum and just inside the region of helium cloud

While the pulse propagates through the neutral gas, it can ionize the Helium in its path, creating  $He^+$ , and can also ionize the  $He^+$ , creating  $He^{++}$ . Ionization in this situation occurs through the quantum phenomena of tunneling ionization (see "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions." for a more complete explanation of tunneling ionization). The computation of tunneling ionization effects is included in the simulation by turning on the tunnelingIonizationFlag parameter in the MCC element of the Region block of the input file.

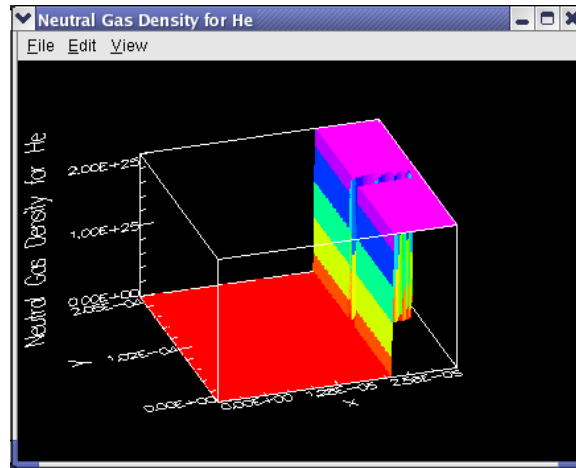


Figure 18: As the laser pulse enters the helium cloud it ionizes all He within its path. The volume within the profile of the laser pulse is essentially void of any neutral He. Tunneling ionization occurs on the edge of the laser pulse where there is a steep gradient in He density.

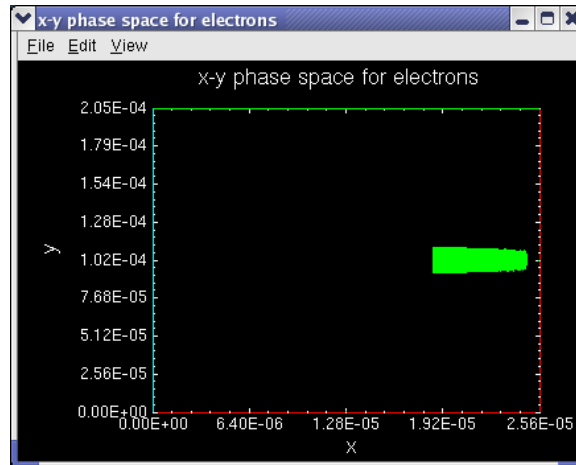


Figure 19: Electrons freed by He and  $He^+$  ionization conform to the profile of the laser pulse.

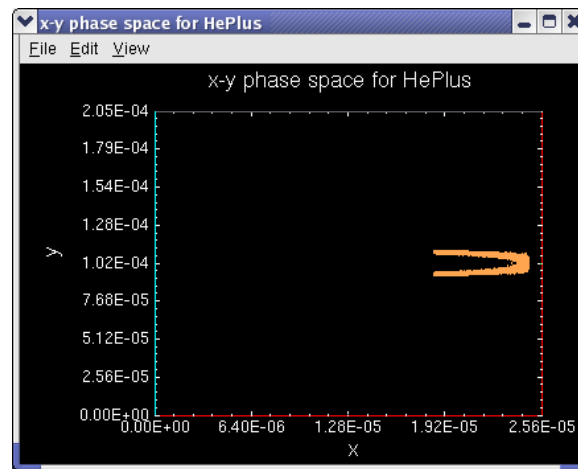


Figure 20: The core of the laser pulse is void of any  $He^+$

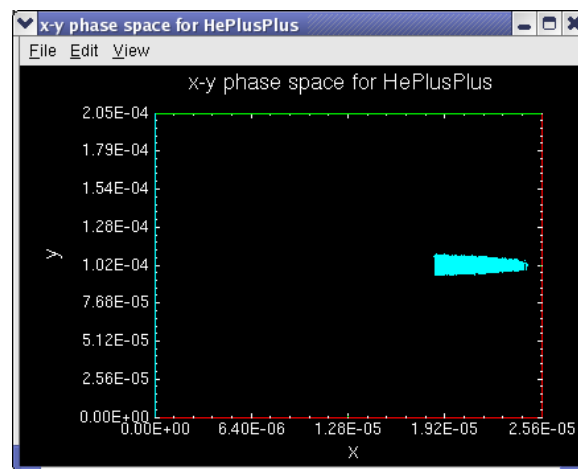


Figure 21:  $He^+$  at the core of the laser pulse is further ionized into  $He^{++}$  since the pulse has sufficient intensity there but not at its edges.

The electron, gas and ion densities shown by this next set of snapshots of the simulation extend, as might be expected, to the point where the laser pulse is about to exit the region of He gas. The pulse has left a trail of fully ionized helium in the form of  $He^+$  and  $He^{++}$ .

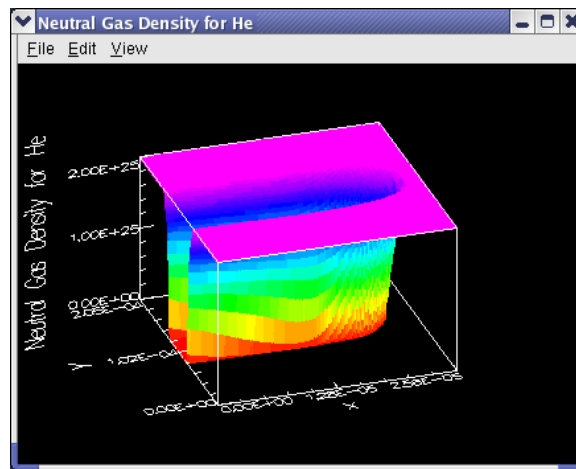


Figure 22: Profile of Helium density in the midst of helium cloud.

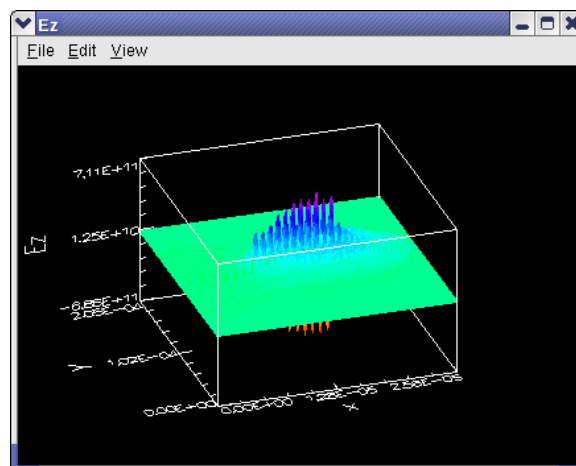
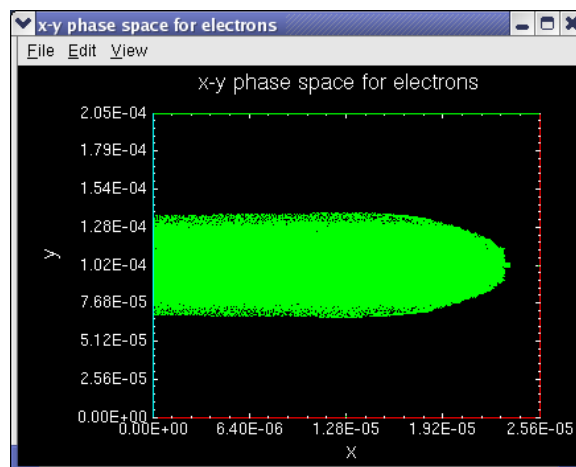


Figure 23: Laser pulse profile in Helium cloud

Figure 24: Electrons freed by He and  $He^+$  ionization. The width of the region where free electrons occur is not only longer due to penetration of the laser pulse but also wider than it was at points near its initial entry.

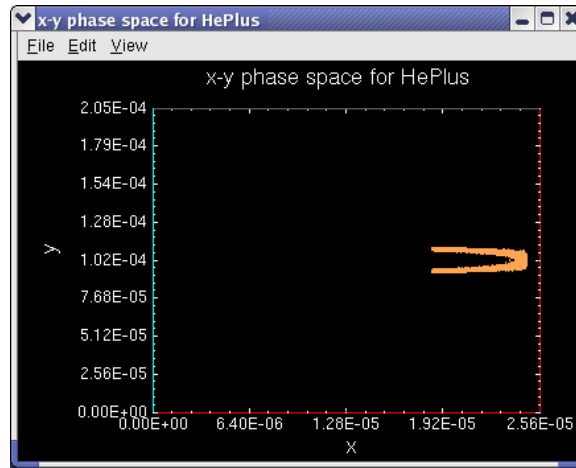


Figure 25:  $He^+$  ion density. As in the earlier snapshot,  $He^+$  is ionized to form  $He^{++}$  in the interior region of the laser pulse.

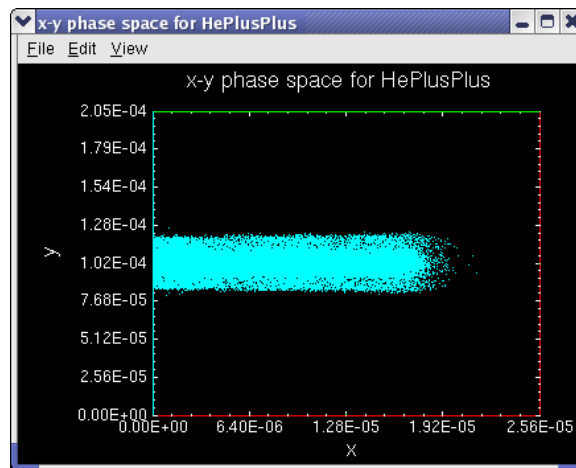


Figure 26:  $He^{++}$  ion density

When the laser pulse exits the helium cloud densities for all species will drop to zero, as shown in Figures 27, 28, 30, and 31. Once the laser pulse has exited the helium cloud completely it propagates through a vacuum once again, as can be seen in Figures 32 and 33.

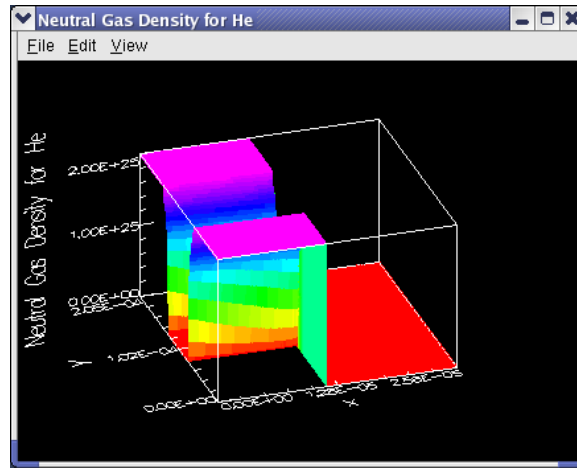


Figure 27: He density profile as laser pulse exits helium cloud.

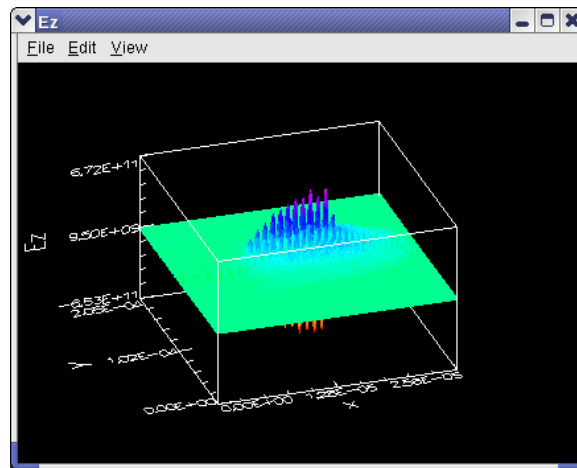


Figure 28: Electric field of laser pulse exiting helium cloud.

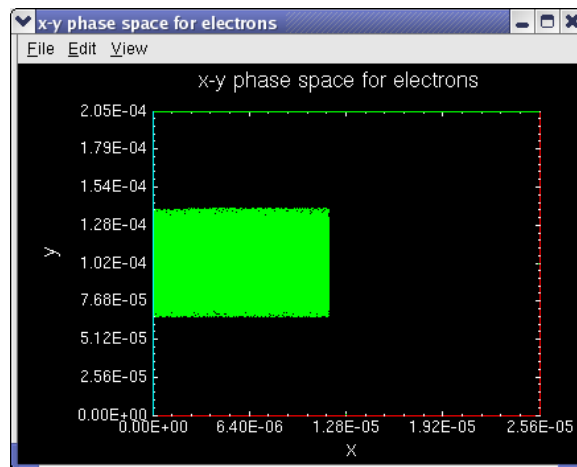


Figure 29: Electron density within helium cloud as laser pulse exits.

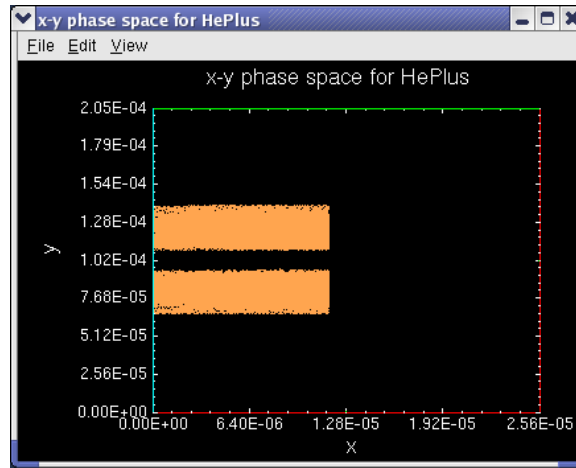
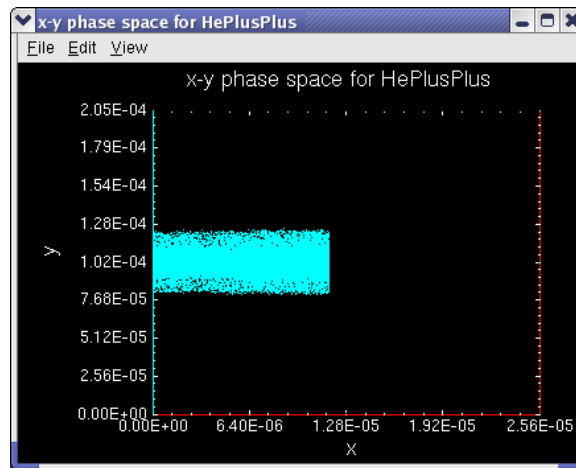
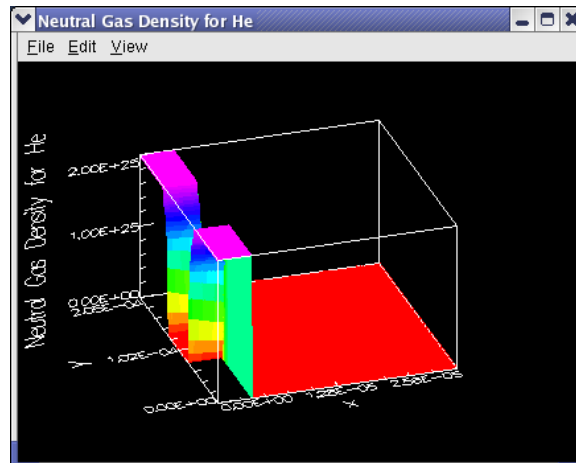
Figure 30:  $He^+$  ion density within helium cloud as laser pulse exits.Figure 31:  $He^{++}$  ion density within helium cloud as laser pulse exits.

Figure 32: Helium density profile (zero) beyond the helium cloud.

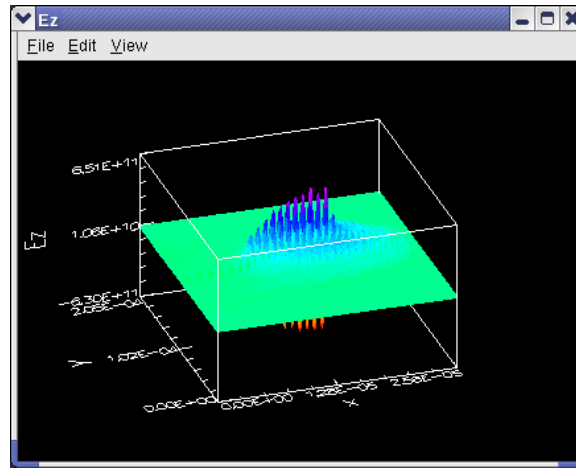


Figure 33: Electric field of laser pulse beyond extent of helium cloud.

### 7.3 Plasma Wakefield Accelerator

This example (`beamplasma.inp`) illustrates the use of OOPIC Pro in modeling a beam-driven plasma wakefield accelerator (PWFA). It is similar to the case discussed in Section V of Particle-in-cell simulations of plasma accelerators and electron-neutral collisions published in *Phys. Rev. Special Topics — Accel. & Beams*, Issue 10 (October 2001).

The input file for this example defines a 2-d simulation conducted in cylindrical coordinates.  $z$  defines an axis of symmetry along which a short pulse of electrons will propagate. The pulse enters a "simulation box" from its left face and propagates to the right. An OOPIC `BeamEmitter` element is defined along part of this boundary, centered on the axis of symmetry, to launch the electron pulse. The remaining boundaries of the simulation box are defined to be perfect `Conductors` except for the section along the  $z$  axis which is established by the `CylindricalAxis` element. The conducting boundaries are far enough away from the electron beam that they have no effect on the phenomena occurring in its vicinity. The simulation box is represented by a single slice through the axis of symmetry and is overlaid with a computational, two-dimensional grid.

The input file calls for the electron beam to have a maximum radial extent on the grid of eight cells. The ratio of the pulse radius to the model radius is four so the computational grid will have 32 cells in the radial direction. Similarly, the ratio of grid extent in the  $z$  direction to that of the radial direction is six so there will be 192 cells along  $z$ .

The spatial extent of the grid is computed from values given for the electron beam characteristics. The rms beam width is given to be  $0.75 \times 10^{-4}$  m and is cut off at 3 times this distance. Using the same ratio of beam radius to the model radius gives a spatial extent in the radial direction of 900 mm. Applying the radial/axial ratio of six again gives an axial extent of 5400 mm.

Geometric spacing on the mesh is chosen to be uniform in both directions so the cells are squares. The time step to be used in the simulation is within limits set by the Courant condition. Even though the Courant condition will guarantee numerical stability of the integration of the field equations, a value arbitrarily smaller than this is chosen to determine the time step. See the input file for this calculation and the scaling factor chosen.

The `BeamEmitter` definition further specifies that in the axial direction the electron pulse will have a Gaussian shape and that it will have energy of 30 GeV. The uniform density of the pre-ionized lithium plasma is set at  $2.1 \times 10^{20} m^{-3}$ . The peak density of the electron beam exceeds that of the background plasma so that the self-fields of the beam will cause "blowout" of the plasma electrons. This will create a wake in the distribution of plasma ions as will be shown.

As in the previous example, a moving window is used to adjust the numerical grid in order to center the physical phenomena of interest as the simulation progresses in time. In this example the geometry of the simulation is initially set up to reflect the simulation box that will be required to fully contain the pulsed electron beam and allow it to propagate within the ionized lithium plasma for a short time. As the pulse approaches the boundaries of the simulation, the boundaries will begin to shift along the direction of propagation in order to keep the pulse within the plasma and more accurately capture the physical phenomena occurring in and around the electron pulse. OOPIC Pro continues to move the simulation box along the direction of pulse propagation.

Figure 34 shows the electron beam pulse shortly after it has been fully launched by the BeamEmitter element.

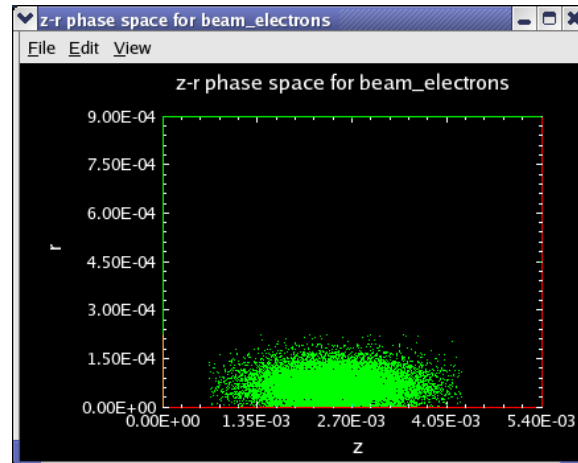


Figure 34: Electron pulse

Figure 35 shows the distribution of plasma electrons ion the presence of the 30 GeV electron pulse as it propagates along the z axis. The wake in the plasma electrons is what drives the EPW. Plasma electrons near the head of the pulse are driven away from the beam but return to the axis near the tail of the beam.

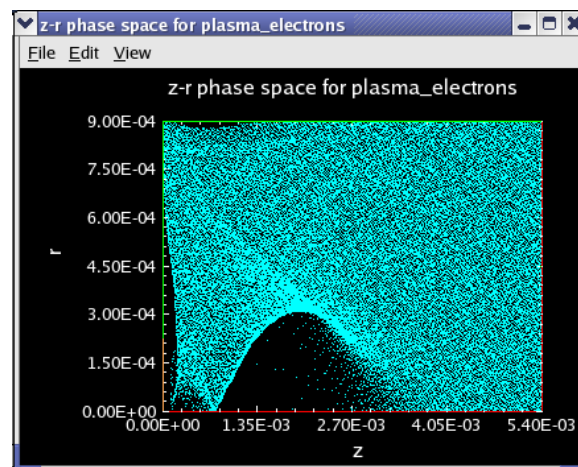


Figure 35: Plasma electron distribution in presence of electron beam showing wake left by electron pulse,  $t = 1.38 \times 10^{-11}$ .

Figure 36 shows the electric field surrounding the electron pulse after the plasma electrons have wrapped around the beam pulse as shown in Figure 35. In the region of the tail of the pulse the density of the plasma electrons has increased by nearly two orders of magnitude from the initial density of  $2.1 \times 10^{20} m^{-3}$ .

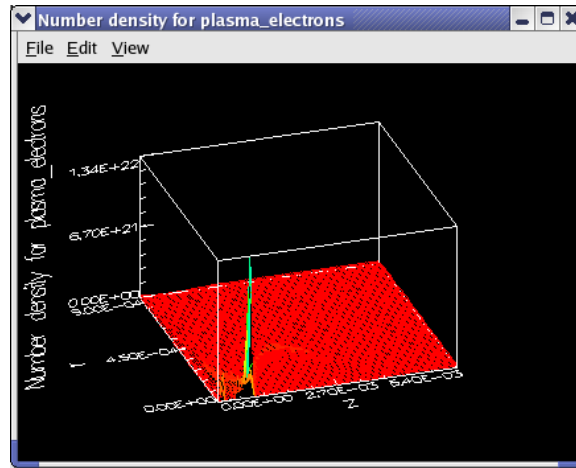


Figure 36: Plasma electron density spikes at the tail of the electron pulse.

Figure 37 shows the Ez component of the electric field within the simulation box with the electron pulse in the same position as Figures 34 through 13.

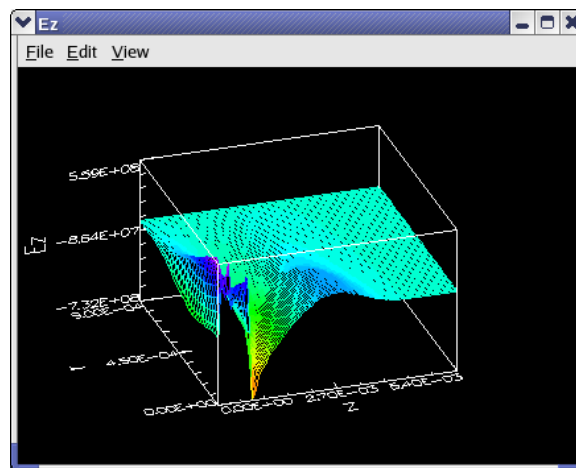


Figure 37: Electric field of electron pulse and surrounding plasma

As the electron beam continues to propagate very little changes except for the expansion of the wake field in the distribution of the plasma electrons. After being blown out of the region along the propagation axis the plasma electrons collapse back toward the axis after the tail of the electron beam has passed. This is shown in Figure 35 as well as, for a later time, in Figure 38. The moving window algorithm in OOPIC Pro has also been invoked by this time.

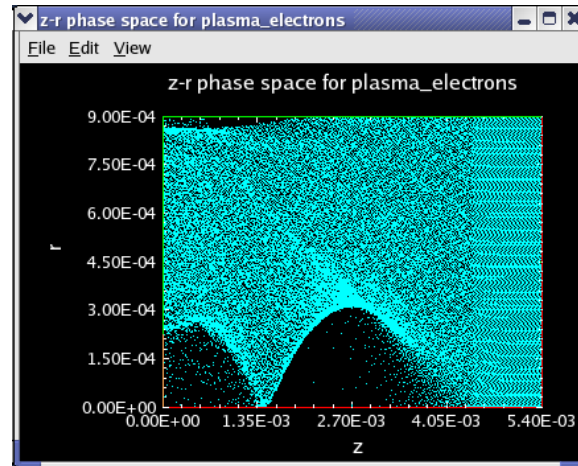


Figure 38: Wake in the plasma electron distribution for time  $t = 2.00 \times 10^{-11}$  sec.

## 7.4 Secondary particle production

This example (`gas.inp`) illustrates the use of OOPIC Pro in modeling ionizing collisions between a streaming beam of electrons and a neutral gas contained in a chamber whose walls are conductors. The gas in this case is argon.

The input file for this example defines a 2-d simulation, staged in cylindrical coordinates.  $z$  defines an axis of symmetry along which an electron beam is directed. Due to symmetry considerations, only one half of a slice through the  $z$ -axis needs to be modeled. The beam enters the chamber from its left face and propagates to the right. An OOPIC `BeamEmitter` element is defined along part of this boundary, centered on the axis of symmetry, to launch the electron beam. The remaining boundaries of the simulation box are defined to be perfect conductors except for the  $z$ -axis, which is established by the `CylindricalAxis` element. The simulation boundaries represent one half of a single slice through the axis of symmetry, overlaid with a computational, two-dimensional grid.

The computational grid is defined to be 10 cells by 10 cells. This grid overlays the physical dimensions which measure 0.1 m horizontally and 0.02 m vertically so that there is a 5:1 aspect ratio for the modeled device. The input file calls for the electron beam emitter element to have a radial extent on the boundary of five grid points, half of the left-most face of the chamber.

An interesting variation on this model will demonstrate the effect of the neutral argon gas. Setting the gas pressure to zero will effectively stream the electron beam into a vacuum. Electrons produced by the ionizing impact are no longer present to drift outside the electron beam. The divergence of the electron beam now occurs within a shorter distance of the `BeamEmitter` source. The quicker divergence occurs because there are no longer any argon ions present to partially cancel the mutual repulsion among the electrons.

## 8 References

1. D.L. Bruhwiler, R.E. Giacone, J.R. Cary, J.P. Verboncoeur, P. Mardahl, E. Esarey, W.P. Leemans and B.A. Shadwick, "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions," *Phys. Rev. ST Accel. Beams* 4, 101302 (2001).
2. D.L. Bruhwiler, D.A. Dimitrov, J.R. Cary, E. Esarey, W.P. Leemans and R.E. Giacone, "Particle-in-cell simulations of tunneling ionization effects in plasma-based accelerators," *Physics of Plasmas* 10 (2003), p. 2022.
3. D.L. Bruhwiler, R.E. Giacone, J.R. Cary, J.P. Verboncoeur, P. Mardahl, E. Esarey, W.P. Leemans and B.A. Shadwick, "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions," [142]*Phys. Rev. Special Topics – Accel. & Beams*, Issue 10 (October 2001). (<http://prst-ab.aps.org/abstract/PRSTAB/v4/i10/e101302>)
4. C. K. Birdsall and A. B. Langdon. *Plasma Physics Via Computer Simulation*. Adam Hilger, 1991.
5. H. Rothard, et al, "Secondary-electron yields from thin foils: A possible probe for the electronic stopping power of heavy ions", *Phys. Rev. A* 41, 2521 (1990).
6. M. A. Furman and M. T. F. Pivi, "Probabilistic model for the simulation of secondary electron emission", *Phys. Rev. ST Accel. Beams* 5, 124404 (2002).

## 9 Advanced Use of OOPIC Pro

This section contains information about using the batch processing capabilities of OOPIC Pro. The OOPIC Pro command line options will be explained.

### 9.1 Command-Line Options

The command line can be used to start an OOPIC Pro run interactively with a GUI or in batch mode without a GUI. A simple example of running OOPIC Pro in batch mode from the command line is:

```
bin/oopicpro -iinput/dring.inp -s100 -nox
```

This command will run the dring (`-iinput/dring.inp`) simulation for 100 steps (`-s100`) without starting the GUI (`-nox`).

Table9.1 is a list of the command line options and their descriptions.

Table 39: Command Line Options in OOPIC Pro

Syntax	Default value	Explanation
-i <inputfile>.inp	No default. Specifying <code>-nox</code> without a valid <code>-i</code> argument results in an error	Specifies the name of the input file containing simulation parameters.
-d <dumpfile>[.h5, dmp]	No default. Specifying <code>-d</code> without an argument results in an error. Default extension is <code>.dmp</code> . The extension is <code>.h5</code> if and only if <code>-h5</code> is specified and <code>-or</code> is not specified.	Specifies the name of the file used for restoring restart data.
-dpn	-dp 0	Specify the number of iterations between dumps. <code>-dp0</code> specifies no intermediate dump.
-nox	If <code>-nox</code> is not specified then run with X interface	If <code>-nox</code> is specified then run without X interface
-sn	-s0	Specify the number of time iterations to be run. <code>-s0</code> specifies no limit on iterations to be executed.
-sf <savefile>[.h5, dmp]	No default. Specifying <code>-sf</code> without an argument results in an error. Default extension is <code>.dmp</code> . The extension is <code>.h5</code> if and only if <code>-h5</code> is specified and <code>-od</code> is not specified	Set the name of the save file. If both the <code>-d</code> and <code>-sf</code> options are set then <code>-d</code> sets the restore file, and <code>-sf</code> sets the save file. If only <code>-d</code> is set then the dumpfile has same filename stub as the restore file.
-h5	Dump and restore using HDF5 The <code>-h5</code> option with or without <code>-or</code> is ignored with regard to restoration of data if a <code>.dmp</code> extension is included in the specification of the restore file by the <code>-d</code> option.	Use HDF file format for dumping and restoring
-h5 -or	-h5 must be set	Restore from hdf5 format file
-h5 -od	-h5 must be set	Dump to hdf5 format file
-display<displayname>	<code>-nox</code> must not be set. <code>\$DISPLAY</code> is default device.	Specify the display device
-exit	Return to GUI control after completing iterations	Exit after completing iterations
-h		Display this list of options
-p<epsfile>[.eps]	No default. Specifying <code>-p</code> without an argument results in an error. The filename extension <code>'eps'</code> is assumed if not specified.	Specify file name for postscript output.
-un	-u1	Update graphics displays every nth iteration.
-id	No incremental dumping.	Incremental dump files will contain a <code>'_i'</code> in the filename followed by the dump index.
-dd	No diagnostic data dump	Dump diagnostics corresponding to H5Diagnostic blocks in input file

## 9.2 Parallel OOPIC Pro (Linux only)

OOPIC Pro has the ability to be run in parallel using the Message Passing Interface (MPI). The implementation of MPI that OOPIC Pro uses is MPICH 1.2.7. More information is available at the MPICH Web site, at <http://www-unix.mcs.anl.gov/mpi/mpich1/>. OOPIC Pro uses a one-dimensional domain decomposition along the x-direction for Cartesian coordinate problems and along the z-direction for Cylindrical coordinate problems. An automatic domain partitioning scheme is used in OOPIC Pro that divides the computational domain equally in each processor based on the number of processors used in a parallel simulation. For an electromagnetic parallel simulation, the same input file that is used in the serial run can be used. However, to run electrostatic simulations in parallel, a change is necessary in the input file — the `ElectrostaticFlag` option needs to be set to use the PETSc Poisson solver, i.e. `ElectrostaticFlag = 6`.

To run the OOPIC Pro in parallel mode, use a command similar to the following:

```
mpirun -np 4 oopicpro -i sample.inp -id -h5 -dd -dp 10 -s 20 -nox
```

The following features are not available when running OOPIC Pro in parallel:

- Splitting of the emitter and plasma source boundaries between processors
- Periodic boundary conditions
- Visualizing of the parallel OOPIC Pro dump results using GUI. Since each processor dumps data available for its local region, the user cannot be able to see the entire computational results together. However, with the `H5Diagnostic` options, the user can merge the dump data from each processor using `h5Merge` commands.
- No smoothing operation can be performed on the charge density calculations.

```
mpirun -np 2 oopicpro -i beamplasmatest.inp -s 20 -dp 20 -h5 -nox
```

where `-np` specifies the number of processes spawned by OOPIC Pro. The number of OOPIC Pro processes may not exceed the number of physical processors present in the host system. `-nox` is a required option under MPI to turn off the GUI operation. This in turn requires that `-i` and an input file name be provided on the `mpirun` command line at a minimum.

If data dumping is requested, separate dump files will be produced by each process. The dump files are distinguished from one another by a `_p` plus the process number (0 to `np-1`) in the file name. The above command would produce the dump files `beamplasmatest_p0.h5` and `beamplasmatest_p1.h5`.

Two executable files are provided with OOPIC Pro to facilitate retrieval of diagnostic data written in HDF5 format. The first of these, `h5Merge`, combines data corresponding to a single diagnostic quantity from the diagnostic dump with positional data in a 2-D array in another HDF5 file. The resulting data can be plotted in three dimensions. This new file can then be converted to text using the `h5dump` utility.

The syntax for `h5Merge` is as follows:

```
h5Merge <readFileStub> <writeFile> <HDF5 path>
```

For example, if the diagnostics from a simulation were dumped to the file `Diagnostics.h5`, then to create a file containing the third component of the electric field at each grid point, one would use the command:

```
h5Merge Diagnostics E3.h5 E3
```

This will result in a HDF5 file being created with 3 columns of data, with each row containing position data and an electric field component, i.e. (x1, x2, E3). This same syntax will also work with `H5Diagnostic` dump files from serial OOPIC runs.

The HDF5 path is the same as the `VarName` keyword in the `H5Diagnostic` block in the input file's `Diagnostics` element within the `Region` block.

Details of the list of `H5Diagnostic` options available are given in Section 6.8.4. Also OOPIC Pro now allows storing of the time history data (such as number of computer particles, average kinetic energy, etc) for the parallel simulations using `H5Diagnostic`. Time history results for the entire computational domain are stored only in the processor #0's `Diagnostics` dump file.

The same `h5Merge` command can also be used for stitching the field variables (such as `Edl`, `EdlBar`, `BdS`, and `I`) that are stored in the dump files resulting from the parallel OOPIC Pro electromagnetic simulations (this option is not available for parallel electrostatic simulations). When stitching the field data from regular dump files, the positions data cannot be stitched to the field vector results. So if the user needs a particular field data or all field data from parallel OOPIC runs, then the user has to set up the `H5Diagnostic` options in the input file.

The syntax for stitching the parallel dump files data is given below:

```
h5Merge <readFileStub> <writeFile> <HDF5 dataset name> 0 0
```

The second-to-last flag in the above command, sets whether the merged file is appended to or created from scratch. (In case one wants to merge multiple fields into the same file). The last flag is whether the data files are OOPIC `H5Diagnostic` files or regular dump files. The default is '1', for `H5Diagnostic` files and it should be '0' for regular dump files.

For example if one wants merge the `BdS` results from the parallel dump files,

```
h5Merge sample_i0 BdSData.h5 /Fields/BdS 0 0
```

In the above command, a complete HDF5 path (`/Fields/BdS`) to the dataset (`BdS`) needs to be given.

Another executable included with OOPIC Pro, `h5PtclMerge`, concatenates particle data from all the particle groups in the dump file(s). The syntax below explains how to merge the particle data from the parallel dump files.

```
h5PtclMerge <readFileStub> <writeFileStub> <HDF5Group> column1 column2
```

In the above command, the `<HDF5Group>` is the name of the species group the user wants to merge. The last two flags refer to the particle column data. The particles data such as positions (x, y in Cartesian coordinates or z, r in cylindrical coordinates) and velocities ( $V_x$ ,  $V_y$ ,  $V_z$  in Cartesian coordinates or  $V_z$ ,  $V_r$ ,  $V_\theta$  in cylindrical coordinates) are stored in the OOPIC dump files in a set of five columns format (x y  $V_x$   $V_y$   $V_z$  for cartesian coordinates or z r  $V_z$   $V_r$   $V_\theta$  for cylindrical coordinates). So one can merge any two set of columns of particle data (by specifying them when running the `h5PtclMerge` executable) from the OOPIC Pro's parallel dump files. For example if one wants to merge the electrons position data from the parallel dump files use the command as below.

```
h5PtclMerge sample_i0 PtclsPosition.h5 /electrons 0 1
```

To get a merged file with ( $y, V_y$  in Cartesian coordinate or  $r, V_r$  in Cylindrical coordinate) results, then the data columns would be changed as 1 and 3 instead of 0 and 1.

```
h5PtclMerge sample_i0 PtclsPosition.h5 /electrons 1 3
```

## 10 FAQ — Frequently Asked Questions

The OOPIC Pro FAQ is a collection of answers to questions that have been asked by users.

### 10.1 Can sputtering (removal of surface atoms due to incident ions) be modeled?

Yes. Please see Section 6.5.6.

### 10.2 Are instabilities in the model more likely related to overly large time step or high np2c?

The time step needs to be small enough so that the fastest species does not cross more than one cell per time step; some users make it slightly smaller.

The cell size needs to be small enough, too. In theory the cell size needs to be smaller than the smallest scale length of the system, so if I were modeling a sheath I would make it half a Debye length at the most. One user of OOPIC Pro uses the following for pseudosparks.

For a hollow cathode size of the order:

- 10 microns — cell size =  $2e-7$
- mm — cell size =  $0.1e-3$
- cm — cell size =  $0.25e-3$  (due to a 5mm anode-cathode gap)

Too large a particle weighting increases fluctuations.

### 10.3 How does OOPIC Pro deal with 3D parameters such as density? In a 2-d simulation, how are these 3D values calculated?

For Cartesian geometry, 3D quantities such as density are measured per unit meter in the ignorable (z) dimension. OOPIC Pro makes the following assumptions:

1. No physical quantity varies along the ignorable (i.e., z) axis, so  $d/dz \rightarrow 0$  in all cases
2. Macro-particles are really lines of charge, not point particles
3. In order to assign definite values to physical characteristics, one imagines that the simulation domain has a 1 m extent along z, with periodic boundary conditions — this is a helpful mental model that takes into account the two rules above
4. Therefore, when trying to interpret the relevance of a simulation to a 3D device, one must interpret quantities such as energy as having units of energy/meter, and then scale the number up or down according to the actual extent (along z) of the physical device.
5. To reconcile items 2 and 3 above, one can imagine that macro-particles are 1 meter long lines of charge, with their mass and charge equally distributed along their length.

When using cylindrical (r-z) geometry in a OOPIC Pro simulation, the situation is slightly more complicated. In this case, the following assumptions are made by OOPIC Pro:

1. No physical quantity varies as a function of the azimuthal angle, so  $d/d_\theta \rightarrow 0$  in all cases
2. Macro-particles are rings of charge, with radius given by their r coordinate
3. In order to assign definite values to physical quantities, one imagines that the simulation domain is rotated around the r=0 axis, so that the total volume is typically  $V = z_{max} * (2 * \pi * r_{max})$ .

#### **10.4 Do the emitters and beamemitters reabsorb electrons or the particles they are emitting (i.e. electrons come back to emitter as a result of manipulating the fields)?**

The emitted electrons respond self-consistently to the fields in the vicinity of the emitter. If these fields drive any electrons back into the emitting surface, then those electrons will be removed from the simulation.

#### **10.5 What are the parameters I need to use to do relativistic simulations in OOPIC Pro?**

For a simulation that includes relativistic particles, you must choose `ElectrostaticFlag = 0` and `NonRelativisticFlag = 0` in the `Control` block. Other choices will most likely result in a simulation giving non-physical results.

## A Sample input files

### A.1 beamplasmatest.inp

```

beamplasmatest
{
High-energy electron bunch enters a quiet plasma in cylindrical geometry --
Low resolution and fewer particles are used here for testing purposes.
This simulation models a beam-plasma wake-field accelerator:
a) The background plasma is pre-ionized. Ions are assumed stationary.
b) Beam density exceeds electron plasma density, so the beam "blows out"
plasma electrons near the symmetry axis.
c) The electron beam is Gaussian in z and (will soon be) Gaussian in r.
d) Electrons at the head of the beam are decelerated by the resulting
electromagnetic fields, while electrons near the tail of the beam
are accelerated by these fields.
e) The electron beam is overfocussed by these fields and so executes
betatron oscillations; however, the focussing force varies axially.
Moving window:
a) Once the electron beam has entered the grid and is close to the far
edge of the simulation region, a moving window algorithm is invoked
so that the beam can be modeled for long times.
Boundary conditions:
a) The simulation region must be bounded by either conductors or
insulators, in order to capture lost particles.
b) Conductors were chosen, to avoid any charge build up.
c) The choice of conducting boundary conditions means that electric
fields parallel to the boundaries are forced to zero; however,
fields near the boundaries of the simulation must be small anyway
to accurately model a semi-infinite plasma, so this is OK.
}
// Define variables that can be used throughout this input file.
Variables
{
// First, define some useful constants.
pi = 3.14159265358979323846
speedOfLight = 2.99792458e+08
electronMass = 9.1093897e-31
unitCharge = electronMass * 1.75881962e11
electronCharge = -1. * unitCharge
electronMassEV = electronMass * speedOfLight * speedOfLight / unitCharge
ionCharge = unitCharge
unitMassMKS = electronMass / 5.48579903e-04
lithiumMassNum = 6.942
lithiumMass = unitMassMKS * lithiumMassNum
// Next, define the parameters of the high-energy electron beam.
beamEnergyEV = 30.0e+09
beamGammaMin1 = beamEnergyEV / electronMassEV
beamGamma = 1 + beamGammaMin1
beamBetaGamma = sqrt( beamGammaMin1 * (beamGammaMin1+2) )
beamBeta = beamBetaGamma / beamGamma
totalNumBeam = 4.0e+10
totalBeamCharge = totalNumBeam * electronCharge
rmsBeamRadius = 0.75e-04
rmsBeamLength = 6.00e-04
rmsBeamTime = rmsBeamLength / speedOfLight
radialCutoffFac = 3
axialCutoffFac = 3
totalBeamRadius = radialCutoffFac * rmsBeamRadius
totalBeamLength = 2 * axialCutoffFac * rmsBeamLength
beamAspectRatio = totalBeamLength / totalBeamRadius
totalBeamArea = pi * totalBeamRadius * totalBeamRadius
rmsBeamVolume = pi * rmsBeamRadius * rmsBeamRadius * rmsBeamLength
rmsEnergySpread = 0.001
beamTempEV = rmsEnergySpread * beamEnergyEV
thermalBeamSpeedEV = 0.5 * beamTempEV
rmsNormalizedEmittance = 4.0e-06
rmsBeamSize = rmsBeamRadius / sqrt(2)
rmsThermalBeta = rmsNormalizedEmittance / rmsBeamSize
rmsThermalGamma = 1. / sqrt(1.-rmsThermalBeta*rmsThermalBeta)
rmsVelocityMKS = rmsThermalBeta * speedOfLight
rmsVelocityEV = (rmsThermalGamma-1.)*electronMassEV
rmsEfactor = 8.0e-04
rmsVfactor = 1.0e-04
// Define the number of grids in R and Z
lengthOverRadiusAspectRatio = 6
simRadiusOverBeamRadius = 4
numRgridsAcrossBeam = 8

```

```

numZgridsAcrossBeam = numRgridsAcrossBeam * beamAspectRatio
numRgrids = numRgridsAcrossBeam * simRadiusOverBeamRadius
numZgrids = numRgrids * lengthOverRadiusAspectRatio
numCells = numRgrids * numZgrids
// Number of beam particles
numBeamPtclsPerCell = 20
numBeamCells = numRgridsAcrossBeam * numZgridsAcrossBeam
numBeamPtcls = numBeamPtclsPerCell * numBeamCells
beamNumRatio = totalNumBeam / numBeamPtcls
// Intermediate calculations for modeling Gaussian shape of the beam.
invSigRsq = 1.0 / ( rmsBeamRadius * rmsBeamRadius )
invSigZsq = 0.5 / ( rmsBeamLength * rmsBeamLength )
invSigTsq = invSigZsq * speedOfLight * speedOfLight
// Calculate the size of the simulation region, grid spacings, time step.
// We are assuming the same grid size in both z and r
maxRadiusMKS = simRadiusOverBeamRadius * totalBeamRadius
rGridSize = maxRadiusMKS / numRgrids
zGridSize = rGridSize
maxLengthMKS = numZgrids * zGridSize
timeStep = 0.41 * rGridSize / speedOfLight
// This is the desired delay time before the moving window algorithm activates.
movingWindowDelay = 0.94 * maxLengthMKS / speedOfLight
// Calculate peak currents for defining emission of the high-energy beam.
peakCurrentDensity = totalBeamCharge * speedOfLight / rmsBeamVolume / sqrt(2.*pi)
peakCurrent = peakCurrentDensity * totalBeamArea
pulseLengthSec = totalBeamLength / speedOfLight
oneHalfPulse = pulseLengthSec / 2.
oneEighthPulse = pulseLengthSec / 8.
threeEighthsPulse = 3.*oneEighthPulse
sevenEighthsPulse = 7.*oneEighthPulse
// Define the plasma density, number of plasma electron macro-particles, etc.
plasmaDensityMKS = 2.1e+20
simulationVolume = pi * maxRadiusMKS * maxRadiusMKS * maxLengthMKS
totalNumPlasma = plasmaDensityMKS * simulationVolume
numPtclsPerCell = 8
numPlasmaPtcls = numPtclsPerCell * numCells
plasmaNumRatio = totalNumPlasma / numPlasmaPtcls
// Define plasma temperature and resulting flux of electrons into the simulation region.
plasmaTempEV = 0.0
thermalSpeed = speedOfLight * sqrt( plasmaTempEV / electronMassEV )
currentFactor = maxRadiusMKS * thermalSpeed * plasmaDensityMKS * electronCharge
endCurrent = currentFactor * maxRadiusMKS * sqrt(pi/2.)
shellCurrent = currentFactor * maxLengthMKS * sqrt(2.*pi)
}
// This simulation has only one "region", which contains grid, all particles, etc.
Region
{
// Define the grid for this region.
Grid
{
// Define number of grids along Z-axis and physical coordinates.
J = numZgrids
xls = 0.0
xlf = maxLengthMKS
nl = 1.0
// Define number of grids along R-axis and physical coordinates.
K = numRgrids
x2s = 0.0
x2f = maxRadiusMKS
n2 = 1.0
}
// Specify "control" parameters for this region
Control
{
// Specify the time step.
dt = timeStep
// Turn on the moving window algorithm.
movingWindow = 1
shiftDelayTime = movingWindowDelay
// Turn on damping for the high-frequency EM fields
emdamping = 0.49
// Turn off the initial Poisson solve
initPoissonSolve = 0
// Use bilinear current weighting
CurrentWeighting=1
}
// Define the beam electrons.
Species
{
name = beam_electrons
m = electronMass

```

```

q = electronCharge
// rmsDiagnosticsFlag = 1
}
// Define the plasma ions.
Species
{
name = plasma_ions
m = lithiumMass
q = ionCharge
}
// Load the plasma ions over the entire simulation region.
Load
{
speciesName = plasma_ions
density = plasmaDensityMKS
x1MinMKS = 0.0
x1MaxMKS = maxLengthMKS
x2MinMKS = 0.0
x2MaxMKS = maxRadiusMKS
// This specifies a static uniform background (no macro-particles).
np2c = 0
}
// Define the plasma electrons.
Species
{
name = plasma_electrons
m = electronMass
q = electronCharge
}
// Load the plasma electrons over the entire simulation region, but
// leave the last dz strip of cells empty, because this strip must
// be handled separately to accomodate the moving window algorithm.
VarWeightLoad
{
speciesName = plasma_electrons
density = plasmaDensityMKS
x1MinMKS = 0.0
x1MaxMKS = maxLengthMKS - zGridSize
x2MinMKS = 0.0
x2MaxMKS = maxRadiusMKS
np2c = 2 * plasmaNumRatio
// Specify a finite plasma temperature (can be zero, of course).
// units = EV
// temperature = plasmaTempEV
v1thermal = thermalSpeed
v2thermal = thermalSpeed
v3thermal = 0.0
// Specify loading that is more uniform than random
LoadMethodFlag = 1
}
// Load the plasma electrons into the last dz strip of cells, which was
// omitted by the load instruction above.
VarWeightLoad
{
// Name this load group "shiftLoad" so that the moving window algorithm
// knows to invoke it every time the simulation window is shifted.
Name = shiftLoad
speciesName = plasma_electrons
density = plasmaDensityMKS
// The fudged values for x1MaxMKS and x2MaxMKS are required, because a
// bug in the load algorithm occasionally puts a randomly loaded macro-
// particle right on the boundary, which then crashes the code.
x1MinMKS = maxLengthMKS - zGridSize
x1MaxMKS = maxLengthMKS - 0.001 * zGridSize
x2MinMKS = 0.0
x2MaxMKS = maxRadiusMKS - 0.001 * rGridSize
np2c = 2 * plasmaNumRatio
// Specify a finite plasma temperature (can be zero, of course).
// units = EV
// temperature = plasmaTempEV
v1thermal = thermalSpeed
v2thermal = thermalSpeed
v3thermal = 0.0
// Specify loading that is more uniform than random
LoadMethodFlag = 1
}
// Define the beam emitter, which introduces the high-energy beam into the
// simulation.
//VarWeightBeamEmitter
BeamEmitter
{

```

```

speciesName = beam_electrons
I = peakCurrent
// Define the 2-D function F(x,t) that specifies beam emission profile.
xtFlag = 3
nIntervals = 32
F=exp(-invSigRsq*x*x)*exp(-invSigTsq*(t-oneHalfPulse)*(t-oneHalfPulse))*step(pulseLengthSec-t)
// Macroparticles are emitted from the left boundary, close to the axis of symmetry.
j1 = 0
j2 = 0
k1 = 0
k2 = numRgridsAcrossBeam
normal = 1
np2c = beamNumRatio
// Emit particles, directed along the Z-axis, with specified energy and temperature.
units = EV
vldrift = beamEnergyEV
vlthermal = rmsEfactor * rmsVelocityEV
v2thermal = rmsVfactor * rmsVelocityEV
// v3thermal = rmsVfactor * rmsVelocityEV
}
// Specify a perfect conductor along the left boundary. This serves as a particle
// boundary condition (catches particles that leave the simulation) and as a
// field boundary condition (E_r is forced to vanish).
Conductor
{
j1 = 0
j2 = 0
k1 = 0
k2 = numRgrids
normal = 1
}
// Specify a perfect conductor along the radial boundary. This serves as a
// particle boundary condition (catches particles that leave the simulation)
// and as a field boundary condition (E_z is forced to vanish).
Conductor
{
j1 = 0
j2 = numZgrids
k1 = numRgrids
k2 = numRgrids
normal = -1
}
// Specify a perfect conductor along the right boundary. This serves as a
// particle boundary condition (catches particles that leave the simulation)
// and as a field boundary condition (E_r is forced to vanish).
Conductor
{
j1 = numZgrids
j2 = numZgrids
k1 = numRgrids
k2 = 0
normal = -1
}
// Define the cylindrical symmetry axis.
CylindricalAxis
{
j1 = 0
j2 = numZgrids
k1 = 0
k2 = 0
normal = 1
}
}

```

## A.2 Two Stream

```

two_stream_ee_es
{
  This is the classic electron/electron two-stream instability, with
  counter-propagating electron beams having velocities much larger than
  their thermal velocity.
  We work in pseudo-1D geometry. It's 2-d Cartesian geometry, but the
  y-direction is only 2 cells across with periodic BC's. The x-direction is
  long.
  The field solve is electrostatic. Hence, there is also a background
  distribution of ions to provide charge neutrality.
  For the theory, see Sections 5-6 and 5-7 of the classic reference C.K.
  Birdsall and A.B. Langdon Plasma Physics via Computer Simulation
  McGraw-Hill, 1985
}
Variables
{
  // General numerical parameters
  PI = 3.14159
  // *****
  // General physical parameters
  // *****
  electronMassMKS = 9.1094e-31
  electronCharge = -1.6022e-19
  speedLight = 2.9979e8
  unitMassMKS = electronMassMKS / 5.48579903e-04
  HeMassNum = 4.0026
  HeMassMKS = unitMassMKS * HeMassNum
  ionDensityMKS = 1.0e+24
  electronDensityMKS = 0.5 * ionDensityMKS
  plasmaFrequency = 2.*PI*9000.*sqrt(2.e-06*electronDensityMKS)
  ds = 0.005 * speedLight / plasmaFrequency
  // *****
  // Grid parameters
  // *****
  Nx = 256
  Ny = 2
  dx = ds
  dy = dx
  Lx = Nx * dx
  Ly = Ny * dy
  simulationVolume = Lx * Ly
  numCells = Nx * Ny
  //
  // streaming electrons
  //
  totalNumElectrons = electronDensityMKS * simulationVolume
  numElectronsPerCell = 4
  numElectronPtcls = numElectronsPerCell * numCells
  electronNumRatio = totalNumElectrons / numElectronPtcls
  rmsElectronSpeedMKS = 3.e+04
  electronVelocityMKS = 1.e+07
  peakCurrentElectrons = electronCharge * electronDensityMKS * Ly * electronVelocityMKS
  //
  // background ions
  //
  totalNumIons = ionDensityMKS * simulationVolume
  numIonsPerCell = 4
  numIonPtcls = numIonsPerCell * numCells
  ionNumRatio = totalNumIons / numIonPtcls
  ionMassMKS = HeMassMKS
  ionCharge = -electronCharge
  rmsIonSpeedMKS = 3000.
  d = 1. / sqrt( 1./(dx*dx) + 1./(dy*dy) )
  timeStep = 0.3 * d / electronVelocityMKS
}
Region
{
  Grid
  {
    J = Nx
    x1s = 0.0
    x1f = Lx
    n1 = 1.0
    K = Ny
    x2s = 0.0
    x2f = Ly
    n2 = 1.0
    Geometry = 1          // 2-d (x-y) slab geometry
  }
}

```

```

PeriodicFlagX1 = 1 // periodic in x1 (x)
PeriodicFlagX2 = 1 // periodic in x2 (y)
}
Control
{
dt = timeStep
// Use the multigrid electrostatic field solve
ElectrostaticFlag = 4
}
Species
{
name = electrons
m = electronMassMKS
q = electronCharge
}
Species
{
name = ions
m = ionMassMKS
q = ionCharge
subcycle = 10
}
// Load the cold, background ions over the entire simulation region
Load
{
speciesName = ions
density = ionDensityMKS
x1MinMKS = 0.0
x1MaxMKS = Lx
x2MinMKS = 0.0
x2MaxMKS = Ly
np2c = ionNumRatio
// specify MKS units for all velocities
units = MKS
v1thermal = rmsIonSpeedMKS
v2thermal = 0.
v3thermal = 0.
}
// Load the right-going plasma electrons over the entire simulation region
Load
{
speciesName = electrons
density = electronDensityMKS
x1MinMKS = 0.0
x1MaxMKS = Lx
x2MinMKS = 0.0
x2MaxMKS = Ly
np2c = electronNumRatio
// specify MKS units for all velocities
units = MKS
vldrifft = electronVelocityMKS
v1thermal = rmsElectronSpeedMKS
v2thermal = 0.
v3thermal = 0.
}
// Load the left-going plasma electrons over the entire simulation region
Load
{
speciesName = electrons
density = electronDensityMKS
x1MinMKS = 0.0
x1MaxMKS = Lx
x2MinMKS = 0.0
x2MaxMKS = Ly
np2c = electronNumRatio
// specify MKS units for all velocities
units = MKS
vldrifft = -electronVelocityMKS
v1thermal = rmsElectronSpeedMKS
v2thermal = 0.
v3thermal = 0.
}
}
}

```

**List of Tables**

1	OOPIC Pro Data Types . . . . .	30
2	Operators Defined in OOPIC Pro . . . . .	31
3	Constants Defined in OOPIC Pro . . . . .	31
4	OOPIC Pro Mathematical Functions . . . . .	31
5	Grid Parameters . . . . .	33
6	Control Element Parameters . . . . .	33
7	Species Parameters . . . . .	39
8	Monte Carlo Collision Parameters . . . . .	40
9	Load Parameters . . . . .	45
10	PlasmaSource Parameters . . . . .	47
11	Grid Spacing Parameters . . . . .	48
12	Grid Spacing MKS Parameters . . . . .	48
13	Species Parameters . . . . .	50
14	Dielectric Parameters . . . . .	51
15	Temperature Rise Diagnostic Parameters . . . . .	52
16	Polarizer Parameters . . . . .	54
17	Secondary Parameters . . . . .	55
18	Secondary2 Parameters . . . . .	56
19	Secondary3 Parameters . . . . .	57
20	Sputter Parameters . . . . .	58
21	CurrentRegion Parameters . . . . .	59
22	ExitPort Parameters . . . . .	61
23	PortGauss Parameters . . . . .	62
24	Generic Emitter Parameters . . . . .	64
25	BeamEmitter Parameters . . . . .	64
27	VarWeightBeamEmitter Parameters . . . . .	65
28	FowlerNordheimEmitter Parameters . . . . .	66

*LIST OF TABLES*

107

29	Temperature Rise Diagnostic Parameters . . . . .	69
30	Diagnostic Geometry Parameters . . . . .	70
31	Diagnostic Parameters . . . . .	70
32	Parameters for Time History of a Point . . . . .	71
33	Time History Diagnostic Parameters . . . . .	72
35	H5Diagnostic Parameters . . . . .	73
36	Diagnostic Bounday Parameters . . . . .	73
37	Other Diagnostic Parameters for Time History . . . . .	74
38	H5Diagnostic - Time History Diagnostics . . . . .	74
39	Command Line Options in OOPIC Pro . . . . .	94

**List of Figures**

1	The input file dialog window. . . . .	14
2	OOPIC Pro Control Window . . . . .	15
3	Example Diagnostics Display Selection List . . . . .	17
4	A 2-D Particle Plot . . . . .	18
5	A 2-D Phase Space Plot . . . . .	19
6	A 2-D Vector Plot . . . . .	19
7	A 3-D surface plot . . . . .	20
8	A 2-D Line Plot . . . . .	20
9	Options Editor window for a 2-d diagnostic plot . . . . .	22
10	Diagram of Time Dependent Envelope Function . . . . .	32
11	Klystron profile . . . . .	77
12	Cross-section of electron beam traversing the klystron . . . . .	78
13	Scaled electron velocity as a function of axial displacement . . . . .	78
14	Poynting vector at the Input Cavity interface . . . . .	79
15	Poynting vector at the Output Cavity interface . . . . .	79
16	Laser pulse profile in vacuum . . . . .	80
17	Profile of Helium density in the area of vacuum and just inside the region of helium cloud . . . . .	81
18	As the laser pulse enters the helium cloud it ionizes all He within its path. The volume within the profile of the laser pulse is essentially void of any neutral He. Tunneling ionization occurs on the edge of the laser pulse where there is a steep gradient in He density. . . . .	82
19	Electrons freed by He and $He^+$ ionization conform to the profile of the laser pulse. . . . .	82
20	The core of the laser pulse is void of any $He^+$ . . . . .	83
21	$He^+$ at the core of the laser pulse is further ionized into $He^{++}$ since the pulse has sufficient intensity there but not at its edges. . . . .	83
22	Profile of Helium density in the midst of helium cloud. . . . .	84
23	Laser pulse profile in Helium cloud . . . . .	84
24	Electrons freed by He and $He^+$ ionization. The width of the region where free electrons occur is not only longer due to penetration of the laser pulse but also wider than it was at points near its initial entry. . . . .	84

25	$He^+$ ion density. As in the earlier snapshot, $He^+$ is ionized to form $He^{++}$ in the interior region of the laser pulse. . . . .	85
26	$He^{++}$ ion density . . . . .	85
27	He density profile as laser pulse exits helium cloud. . . . .	86
28	Electric field of laser pulse exiting helium cloud. . . . .	86
29	Electron density within helium cloud as laser pulse exits. . . . .	86
30	$He^+$ ion density within helium cloud as laser pulse exits. . . . .	87
31	$He^{++}$ ion density within helium cloud as laser pulse exits. . . . .	87
32	Helium density profile (zero) beyond the helium cloud. . . . .	87
33	Electric field of laser pulse beyond extent of helium cloud. . . . .	88
34	Electron pulse . . . . .	89
35	Plasma electron distribution in presence of electron beam showing wake left by electron pulse, $t = 1.38 \times 10^{-11}$ . . . . .	89
36	Plasma electron density spikes at the tail of the electron pulse. . . . .	90
37	Electric field of electron pulse and surrounding plasma . . . . .	90
38	Wake in the plasma electron distribution for time $t = 2.00 \times 10^{-11}$ sec. . . . .	91

## Index

- A, 50
- a0, 50
- a1, 50
- AFN, 66
- Algebra, 75
- Algebra rule, 75
- alternating direction implicit, 36
- amp\_p0, 62
- analyticF, 40, 45, 47
- Apple, 11
- arithmetic operators, 31
- automatic saves, 23
- Ave, 71
  
- B01, 34
- B01analytic, 34
- B02, 34
- B02analytic, 34
- B03, 34
- B03analytic, 34
- B1, 72
- B1init, 34
- B2, 72
- B2init, 34
- B3, 72
- B3init, 34
- batch mode, 93
- BeamEmitter, 27, 64, 77
- betaFN, 66
- Bf, 34
- BFN, 66
- Boundary Conditions, 26
- boundary conditions, 47
  
- C, 50
- charge density, 35
- CHARGE\_EXCHANGE, 43
- chirp\_p0, 62
- clipboard, 21, 22
- collisionFlag, 40
- collisionModel, 39
- Comb, 71
- command line, 93
- Conductor, 26, 51, 80
- conjugate gradient, 36
- constants, 31
- Control, 26, 33
- convention, 18
- Coronal model, 36
- current region, 59
- CurrentRegion, 27, 59
  
- CurrentWeighting, 34
- CvFN, 66
- CyFN, 66
- CylindricalAxis, 27, 60
  
- Densities, 68
- density, 45
- Description block, 25
- diagnostic boundaries, 73
- diagnostic plot
  - 2-D, 17
    - crossHair, 21
    - data point, value, 21
    - zooming, 21
  - 2-D line, 20
  - 3-D, 19, 22
    - grid, 23
    - inverting black and white, 23
    - shading, 23
    - wire frame, 23
  - axis, 21
  - titles, 22
- diagnostic plots, 17
  - manipulating, 20
  - vector, 19
- diagnostics, 67
- diagnostics, defining, 70
- Dielectric, 26, 51
- DielectricRegion, 27, 58
- DielectricTriangle, 27, 59
- Divergence Error, 68
- divergence error, 68
- DivergenceCleanFlag, 34
- dt, 33
- dump file, 15
- dump file, loading, 24
- dump files, 23
  
- e, 31
- E1, 72
- E1init, 34
- E2, 72
- E2init, 34
- E3, 72
- E3init, 34
- ecxFactor, 40
- Eemit, 55
- EFFlag, 50
- EfieldFrequency, 40
- elastic, 42
- ElecCollXSectionFile, 42

- electron beam, 90
- electrostatic potential, 35
- ElectroStaticFlag, 35
- ElectrostaticFlag, 34
- emdamping, 33
- EmitPort, 27, 65
- energy\_max, 50
- energy\_min, 50
- energyScaleFactor, 52, 69
- Equipotential, 27, 53
- er, 51
- error log, 16
- eSpecies, 40
- ETIPolarizationFlag, 40
- evaluation version, 13
- excitation, 42
- ExitPort, 27, 61
- ExitPorts, 80
  
- F, 50
- fill, 50
- focus\_p0, 62
- Fowler-Nordheim, 27, 66
- FowlerNordheimEmitter, 27, 66
- Fransseed, 34
- frequency, 50
  
- Gap, 27, 61
- gas, 40
- gasOffTime, 33
- Generalized Minimal Residual, 36
- Generic Emitter Parameters, 63
- Graphical User Interface
  - buttons, 16
  - control window, 14
  - Menus, 15
- Grid, 26, 32
- Grid and Control, 26
- GUI, *see* Graphical User Interface
  
- HDF5, 23, 73, 94
- heatFluxParticlesFlag, 52, 69
- helium cloud, 80
- HistMax, 71
  
- I, 64
- I1, 72
- I2, 72
- I3, 72
- icxFactor, 40
- IdiagFlag, 50
- Ihist\_avg, 50
- Ihist\_ien, 50
- Iloop, 27, 60
- image sequences, *see* movies
  
- InfiniteBFlag, 34
- initPoissonSolve, 34
- input file, 25
  - control parameters, 33
  - description block, 25
  - example, 27
  - functions, 31
  - grid, 32
- input files
  - syntax, 30
- input files, examples, 14
- installation, 13
- intEd11, 72
- intEd12, 72
- intEd13, 72
- integral, 71
- IonCollXSectionFile, 43
- ION\_ELASTIC, 43
- ionization, 42
- ionized lithium plasma, 89
- ionzFraction\_i, 40
- iSpecies, 40, 55
- iSpeciesPlusPlus, 40
  
- j1BeamDump, 34
- j2BeamDump, 34
  
- klystron, 77
  
- laser wavelength, 80
- laser/gas interaction, 81
- LaserSpotSize, 72
- Lcutoff, 45, 64
- Lcutoff1, 47
- Lcutoff2, 47
- Limit, 74
- limit rule, 74
- Linux, 11
- Load, 26, 44
- LoadMethodFlag, 45
  
- m, 39
- Macintosh, 11
- MarderIter, 34
- MarderParameter, 34
- materialDensity, 52, 69
- materialRange, 52, 69
- Maxwellian velocity distributions, 26
- MCC, 26, 39
- Mesa graphics library, 24
- Microsoft Windows, 11
- modeling ionizing collisions, 91
- Monte Carlo collision, 26
- Monte Carlo collisions, 26, 39
- movies, 24

- moving window, 80, 89, 90
- movingWindow, 33
- Multigrid, 36
- name, 45, 50
- NElectronCollisions, 42
- nEmit, 65
- nenergybins, 50
- nfft, 71
- nIntervals, 64, 66
- NIonCollisions, 43
- NonRelativisticFlag, 34
- normal, 50
- notation, 30
- np2c, 38, 45, 47, 64
- np2cFactor, 34
- nSmoothing, 34
- nthetabins, 50
- numMacroParticlesPerCell, 40
- nxbins, 50
- offset, 62
- OS X, 11
- output log, 16
- particle creation, 38
- particle distributions, 38
- particle emitters, 27, 63
- particleLimit, 34, 39
- PETSc, 36
- PetscKSPTType, 34
- phase, 50
- phase space plots, 18
- Phi, 54, 68
- phi, 72
- Phi\_wFN, 66
- PI, 31
- PIC, 11
- Plasma Radiation, 36
- plasma source, 46
- plasma wakefield accelerator, 88
- PlasmaRadiationFlag, 36
- PlasmaSource, 26, 46
- plots, 17, *see* diagnostic plots
- polarizationEB, 71
- Polarizer, 27, 54
- polarizer, 54
- PortGauss, 27, 80
- PortGuass, 62
- Ports, 27, 60
- ports
  - ExitPort, 61
- Poynting Vector, 68
- Poynting vector, 79
- poynting1, 72
- poynting2, 72
- poynting3, 72
- presidue, 34
- pressure, 40
- psd1dFlag, 71
- pullLeng\_p0, 62
- pulse launched into a vacuum, 81
- pulShp\_p0, 62
- Q, 72
- q, 39
- QuseFlag, 51
- reflection, 51
- refMaxE, 51
- Region, 26, 47
- Relation, 75
- Relation rule, 75
- relativisticMCC, 40
- Restart, 16
- Rho, 68, 72
- rmsDiagnosticsFlag, 39
- Run button, 16
- Secondary, 27, 55
- secondary, 55
- Secondary particle production, 91
- secondary species, 55
- Secondary2, 27, 55
- Secondary3, 56
- secSpecies, 55
- Segment, 48
- shiftDelayTime, 33
- simulation box, 89
- sourceRate, 47
- Species, 26, 38
- speciesName, 45, 64, 72
- speciesName1, 47
- speciesName2, 47
- specificHeat, 52, 69
- Specifying External Magnetic Fields, 37
- spotSize\_p0, 62
- Sputter, 57
- Step button, 16
- Stop control, 16
- subcycle, 39
- Synchrotron Radiation, 34
- tdelay, 50
- tdelay\_p0, 62
- temperature, 40, 45, 64
- Temperature rise at the boundaries, 52, 69
- temperature1, 47
- temperature2, 47
- TERM, 13

- text dump, 21, 22
- tfall, 50
- thetadot, 64
- theta\_max, 50
- theta\_min, 50
- threshold, 39, 55
- Time History of a Line, 71
- Time History of a Point, 71
- time, simulation, 15
- title, 71
- tpulse, 50
- trace, 21
- transmissivity, 54
- transparency, 50
- trise, 50
- tunnelingIonizationFlag, 40
  
- Ucutoff, 45, 64
- Ucutoff1, 47
- Ucutoff2, 47
- units, 45, 64
- units1, 47
- units2, 47
- User-Defined Cross Section, 41
- User-Defined Diagnostics, 27
  
- v1drift, 45, 64
- v1drift1, 47
- v1drift2, 47
- v1Lcutoff, 45, 64
- v1Lcutoff1, 47
- v1Lcutoff2, 47
- v1thermal, 45, 64
- v1thermal1, 47
- v1thermal2, 47
- v1Ucutoff, 45, 64
- v1Ucutoff1, 47
- v1Ucutoff2, 47
- v2drift, 45, 64
- v2drift1, 47
- v2drift2, 47
- v2Lcutoff, 45, 64
- v2Lcutoff1, 47
- v2Lcutoff2, 47
- v2thermal, 45, 64
- v2thermal1, 47
- v2thermal2, 47
- v2Ucutoff, 45, 64
- v2Ucutoff1, 47
- v2Ucutoff2, 47
- v3drift, 45, 64
- v3drift1, 47
- v3drift2, 47
- v3Lcutoff, 45, 64
- v3Lcutoff1, 47
- v3Lcutoff2, 47
- v3thermal, 45, 64
- v3thermal1, 47
- v3thermal2, 47
- v3Ucutoff, 45, 64
- v3Ucutoff1, 47
- v3Ucutoff2, 47
- variables, 25
- VarName, 71
- VarWeightBeamEmitter, 27
- VarWeightEmitter, 65
- VarWeightLoad, 26, 46
- Vaughan-based secondary production, 55
- velocity distributions, 44
  
- WaveDirDiagnostic, 72
- waveLeng\_p0, 62
- windowName, 71
  
- x1\_Label, 71
- x1MaxMKS, 40, 45
- x1MinMKS, 40, 45
- x2\_Label, 71
- x2MaxMKS, 40, 45
- x2MinMKS, 40, 45
- x3\_Label, 71
- XOOPIC, 7
- xtFlag, 32, 50