# KDetSim User Guide

| | | |
|---|---|---|
| *Authors* | Liza Zoubakina (liza.zoubakina@mail.utoronto.ca) | Jason Yuan (jason.yuan@mail.utoronto.ca) |
| *Developers and Contributors* | Dr. Gregor Kramberger (gregor.kramberger@ijs.si) | Dr. Daniel Pitzl (daniel.pitzl@desy.de) |

## Introduction

This is a rudimentary guide to the usage and implementation of KDetSim. KDetSim is a custom simulation tool developed by Dr. Gregor Kramberger.

The simulator derives many of its classes and analysis applications from ROOT, a scientific software toolkit developed and released by CERN—the simulator can be thought of as a library for ROOT and extends the capabilities of ROOT. The coding involved in KDetSim is in C++—similar to the coding used in ROOT—and the simulator is meant to be an easier and more user-friendly tool for creating particle detector simulations. Using the KDetSim classes and some C++ coding, users can simulate a variety of different particle detector simulations.

For more information on the classes and models that are used by the software, refer to the “Manuals and Documentations” tab of the KDetSim website (http://kdetsim.org/).

## Table of Contents

# Installation

This guide covers three different methods of installation for the following computer platforms:

- Mac
- Windows with Linux Subsystem
- Linux

It is required to download and install ROOT (Version 6) beforehand. If you have already installed ROOT, then you can skip ahead to the installation instructions for KDetSim.

## Linux

This first section will address the installation of ROOT. If ROOT has been installed, feel free to skip ahead to the KDetSim installation below.

When installing the ROOT simulation software, the user has two options: build from source and a binary installation. A binary installation is the simplest and includes most of the common analysis features that a user would require. For users who want more control over ROOT, an installation from the source file would be recommended. You can find the source and binary packages on the CERN ROOT website (https://root.cern.ch/downloading-root). For the purpose of this manual, we will be using a binary distribution of ROOT.

To install the binary version of ROOT, go to the downloads page for the appropriate version of ROOT and choose the corresponding file for your platform under the "Binary Distributions" header. Open the downloaded package and follow the instructions given in the package's "README" file.

It is very important that ROOT is properly installed in your system before proceeding forward.

Once you have ROOT installed, proceed to install KDetSim. There are two package versions of KDetSim available for installation. One of these versions can be found on the KDetSim website (http://kdetsim.org/KDetSim-Downloads.html) and the other can be sourced from GitHub (https://github.com/pitzl/KDetSim). The instructions provided on the KDetSim website should be sufficient for installing that version of the package. This manual will cover the installation from the GitHub source.

1. Open a terminal window in your Linux system—the following instructions will all be done from your terminal window
2. Clone or download KDetSim from GitHub (use the command "git clone https://github.com/pitzl/KDetSim") and extract the file to a chosen folder
3. In the KDetSim folder, create two new folders: "obj" and "lib"
4. In the Makefile, change the version of the C++ compiler from 17 to 11 (you'll find this near the top of the file)
5. Now, build the package by executing the "make" command
6. Assuming the package has built successfully (no make errors), proceed to the "lib" folder of your installation of ROOT
7. Create a symbolic link by executing the following in your terminal window:
   *ln -s /home/YOU/KDetSim/src/KDetSimDictUX_rdict.pcm   KDetSimDictUX_rdict.pcm*
8. Go to the "examples" folder of your KDetSim installation and open ROOT

9. In ROOT, type “gsystem->Load(“YOUR_KDETSIM_DIRECTORY/lib/KDetSim.sl”);
10. Try executing an example with “.x Pixel_1.C”

For more instructions, refer to the “README” file that can be found in the KDetSim folder or on the GitHub page.

The following will outline some problems that you may encounter with your KDetSim installation and the potential fixes.

- ***Example files don’t run***: This is most likely a problem with the actual example file. Take a look at the output of the example file in your terminal window and see if it points to any errors in the example file. If the output points out syntax errors in the example files, make the edits and try to run the example file again
- ***The “gsystem” command fails***: A possibility for this error could be where you linked the resulting KDetSimDictUX_rdict.pcm file. Make sure that you have linked it to the ROOT library—if the ROOT system is built from source, you will need to link the file to the folder where you have **installed** ROOT and the “lib” folder of the source. Also, try linking the KDetSimDictUX_rdict.pcm file to the “lib” folder in the KDetSim directory

If the above suggested fixes do not work, you may have to reinstall either ROOT, KDetSim or both. When reinstalling, be sure to follow the instructions given by the respective sources as closely as possible.

## Mac

Since the MacOS system shares some similarities with the Linux operating system, the instructions from above will apply to the installation of ROOT and KDetSim on Mac. Again, to follow any procedures that are found on the websites where you obtain your distributions from.

## Windows 10 with Linux Subsystem

Unfortunately, ROOT is not compatible with Windows 10 software, therefore it is required to install a Linux Subsystem beforehand. The steps are below:

1. Enable and install Linux Subsystem on Windows 10 (Ubuntu v. 18.04 recommended)
    a. https://developerinsider.co/stepwise-guide-to-enable-windows-10-subsystem-for-linux/
2. Install ROOT (Version 6 or higher). Make sure you follow ALL of the instructions on the website.
    a. https://medium.com/@blake.leverington/installing-cern-root-under-windows-10-with-subsystem-for-linux-beta-75295defc6d4
    b. Download and install the related graphics software that the website refers to (X11).

It may take several hours for your computer to build and compile ROOT. It is very important that ROOT is properly installed in your system before proceeding forward. To test out the X11 Graphics software, type in the following command into the Ubuntu terminal: *$ xeyes* .

Once you have ROOT installed, proceed to install KDetSim. There are two package versions of KDetSim available for installation. One of these versions can be found on the KDetSim website (http://kdetsim.org/KDetSim-Downloads.html) and the other can be sourced from GitHub (https://github.com/pitzl/KDetSim). The instructions provided on the KDetSim website should be sufficient for installing that version of the package. This manual will cover the installation from the GitHub source.

1. Clone or download KDetSim from the GitHub site. Type the following command into the Ubuntu terminal (in the Home directory).
    a. *$ git clone [https://github.com/pitzl/KDetSim](https://github.com/pitzl/KDetSim)*
2. Follow the steps on the KDetSim readme file located on the GitHub site. In the KDetSim folder, create two new folders called objective and library.
    a. *$ mkdir obj and $ mkdir lib*
3. In the Makefile, change the version of the C++ compiler from 17 to 11 (you'll find this near the top of the file). You can do this via text editor.
4. Now, build the package by executing the "make" command while in the KDetSim folder.
    a. *$ make*
5. Assuming the package has built successfully (no errors), change directory to the "lib" folder of your installation of ROOT.
    a. *$ cd /home/YOU/ROOT/lib*
6. Create a symbolic link by executing the following in your terminal window:
    a. *$ ln -s /home/YOU/KDetSim/src/KDetSimDictUX_rdict.pcm   KDetSimDictUX_rdict.pcm*
7. Go to the "examples" folder of your KDetSim installation and open ROOT
    a. *$ root*
    b. Next, type "*gsystem->Load("YOUR_KDETSIM_DIRECTORY/lib/KDetSim.sl*");
    c. Try executing an example with *".x Pixel_1.C*"

You should see several graphs and a 3D picture of a cube when executing the above command. If so, congratulations! You have successfully downloaded both ROOT and KDetSim. If not, please consider doing the following steps. The following will outline some problems that you may encounter with your KDetSim installation and the potential fixes.

- ***Example files don't run***: This is most likely a problem with the actual example file. Take a look at the output of the example file in your terminal window and see if it points to any errors in the example file. If the output points out syntax errors in the example files, make the edits and try to run the example file again
- ***The "gsystem" command fails***: A possibility for this error could be where you linked the resulting KDetSimDictUX_rdict.pcm file. Make sure that you have linked it to the ROOT library—if the ROOT system is built from source, you will need to link the file to the folder where you have **installed** ROOT and the "lib" folder of the source. Also, try linking the KDetSimDictUX_rdict.pcm file to the "lib" folder in the KDetSim directory

If the above suggested fixes do not work, you may have to reinstall either ROOT, KDetSim or both. When reinstalling, be sure to follow the instructions given by the respective sources as closely as possible.

## Overview of Classes

KDetSim is a program based on the C++ programming language and consists of two main file types: **source** files (src folder) and **header** files (inc folder). It is recommended to go over the basic syntax of C++—specifically the implementation of classes—to understand how these various files, classes, and functions are implemented.

Whenever there is an edit made to any of the source/header files, remember to enter the *"$ make*" command into the terminal while in the KDetSim folder for those changes to take effect. Otherwise, there will be no change to how KDetSim executes your files.

If more information is needed, please refer to the class hierarchy that can be found on the KDetSim website. Alternatively, you can contact the developers and contributors for more information.

### KDetector Class

NOTE: DESCRIPTION IS VAGUE BECAUSE CLASS DESCRIPTION IS UNAVAILBLE)

This is the general detection class. All other detector classes such as KPad, K3D and KPixel are derived from this main class. The usage of KDetector is very open and depends on what the user requires. As this class has access to member functions of KMaterial and KGeometry, a user can define very specific detector setups. For simplicity, the use of the other detector classes may also suit a user's needs. The main difference that exists between the multiple detector classes is the ease for which a detector geometry is initialized. In the KDetector class, several components must be initialized manually—for instance the material and electrodes. In contrast, classes like KPad and K3D have methods such as "K3D::SetUpMaterial(mat)" which will initialize material histograms and additional methods to initialize electrode geometries.

The detector class contains some physical formulas that can be found on the KDetSim website which describes how the simulator simulates certain processes.

For usage of these classes, referring to the example files will be a helpful place to start.

### KMaterial Class

This class defines the major types of materials that are used in the simulator software. In the simulator, there are five major materials that are available for use: silicon, silicon oxide, air, aluminum, and diamond. Each material can be used to define the detector material—the material that is being subjected to the ion irradiation and not the actual material of the detector. In KMaterial, the user can control variables "Mat", "Mobility" and the function "KMaterial::Perm(mat)" to manipulate the material and mobility model that particles entering the material follow.

Change KMaterial variables when you need to set up different types of interfaces for your detector set up.

### KGeometry Class

This class describes the geometry of the volume set up of the container that houses all the other geometries of the system. In KGeometry, the user has access to functions that will define the boundaries of the box that houses the electrode as well as access to functions like "KGeometry::ElRectangle(…)" and "KGeometry::ElCylinder(…)" which create the electrodes.

While there are no variables in this particular class implementation that has a large impact on the transmission of the ion in the detector material.

(MORE DETAILS ARE NEEDED TO COMPLETE THE EXPLANATION OF THIS CLASS DESCRIPTION)

### KField Class

This class is a description of the field properties that are being applied to the simulator. In the class, the functions that are being used define the electric, magnetic and weighting fields as well as Mobility functions to replicate the movement of ions in the material.

Similar to KGeometry, there are not variables for the user to change in order to set up the simulation. Most of the drift and field set ups have already been defined by the developers in the KDetector Class.

SUGGESTION: A DESCRIPTION OF EACH CLASS WOULD BE VERY HELPFUL FOR A BEGINNING USER TO FIGURE OUT HOW EACH CLASS WORKS WITH ANOTHER, AND TO IDENTIFY SOME OF THE CONSTANTS AND PHYSICS FORMULAE THAT ARE BEING USED IN THE SIMULATOR

## How to Construct an Example File

There are several methods in constructing an example file to model various types of detectors. In this guide, we will go over a few of the basic types, which are:

- KDetector example files
- K3D example files

In general, the method of constructing an example file is similar across the detectors, however there are a few differences. In this section, we will also include snapshots of code which demonstrate how an example file is written and what it will generate.

## K3D Example Files

K3D example files allow for greater creativity as that you can build a variety of 3D electrode set-ups. Here are the following steps in constructing a K3D file:

1. Initializing the Parameters
2. Initializing Space Charge and Doping Concentration
3. Constructing the Electrodes
4. Calculating the Fields
5. Initializing the Entry/Exit Points
6. Constructing the Graphs

The last step is of greatest difficulty and forms the bulk of the calculations. There are several pre-set graphs that KDetSim already provides to the user that provide information on basic properties of the simulation (such as current, electric field, and illustration of drift).

### *1. Initializing the Parameters*

First, you need to set up the parameters of your simulation. There are several mandatory parameters that need to be specified and optional parameters that can be added if they are relevant to the experiment.  At the beginning of the document, one sets up the GStyle function to the type of Canvas that they prefer. This function allows for the user to graph several "canvases", which are essentially windows of data that you want to create.

Next, you set up the main **K3D detector** function by creating and initializing a new K3D function. K3D( number of electrodes, x-length of detector in um, y-length, z-length)

You can initialize the voltage of the High-Voltage electrodes by det->Voltage = 200V.
SetUpVolume determines the mesh size and speed of charge. The higher the numbers, the faster your simulation will take due to the increased voxel size.

The material of the electrodes and detectors can be set up by inputting the material number into the relevant functions. Here are the following available parameters: 1 – Silicon, 10 – Diamond.

Next, one can initialize the boundary conditions of the detector. By default, the boundaries are reflective.

Diffusion of charge can be turned on (1) or off (0) through the det->diff function.

```
{

gStyle->SetCanvasPreferGL(kTRUE);

K3D *det=new K3D(1,200,200,100);

det->Voltage=200;
det->SetUpVolume(2,2);

det->SetUpElectrodes(10);
det->SetUpMaterial(10);

det->SetBoundaryConditions();
det->diff=1;
```

### 2. *Initializing Space Charge and Doping Concentration*

If the detector material contains doping concentration, such as silicon or GaAs, the user can input this concentration and relevant space charge that it creates through the following TF3 function.

By declaring a new TF3 histogram, one can construct a space charge histogram and indicate the level of doping concentration. By default, the doping concentration is set to zero. SetParameter determines the type of doping, and currently it is at zero.

```
TF3 *f2=new
TF3("f2","x[0]*x[1]*x[2]*0+[0]",0,3000,
0,3000,0,3000);

f2->SetParameter(0,-2);

det->NeffF=f2;
```

### 3. *Constructing the Electrodes*

This step requires more time and practice, but it becomes easier over time. Unlike the Pixel example files, the user has to manually construct and set-up each individual electrode in K3D. This requires more attention to detail and effort, but also allows for more creativity. This is an example for constructing a single rectangular electrode, you can construct additional electrodes by repeating this process.

To construct an electrode, the user needs to first declare the position and size of the electrode. This is done by creating two floats. Pos[3] = {x-center, y-center, z-center} , Size[3] = {x-radius, y-radius, z-radius}. The radius here is the extension of length on either side of the centre point.

Then, one needs to write the ElRectangle function (two floats, type of electrode, material of electrode).

Type of electrode parameters: 0 – No electrode, 1 – Ground electrode, 2 – High Voltage electrode, 16385 – Collector electrode at GND.

```
Float_t Pos[3]={100,100,98};

Float_t Size[3]={100,100,2};

det->ElRectangle(Pos,Size,0,20);
```

### 4. *Calculating the Fields*

Finally, once all of the parameters have been initialized, it is possible to calculate the electric and Ramos fields by writing the following statements.

These KDetector functions calculate the corresponding fields. Parameters: 0 – Electric field, 1 – Ramo weighting field.

```
det->CalField(0); // E field

det->CalField(1); // Ramo weighting
potential
```

### 5. *Initializing the Entry/Exit Points of Ion Injection*

To simulate the path of a single ion injection into the detector material, the user can initialize the entry/exit points by entering the following parameters. To simulate multiple entry/exit points, the user can move onto step 6, which can loop through x, y, and z parameters.

```
det->enp[0]=50; //entry point for x
det->enp[1]=50; //entry point for y
det->enp[2]=1;  //entry point for z

det->exp[0]=50; //exit point for x
det->exp[1]=50; //exit point for y
det->exp[2]=100; //exit point for z
```

### 6. *Constructing the Graphs*

The last step requires the user to construct  the necessary code and functions to generate and draw the relevant histograms/graphs. In KDetSim, there are several pre-set graphs that generate a variety of essential information.

Before creating the graph, it is necessary to declare a new TCanvas variable (which opens a graph window once the simulation is completed). The user can then set the title and input the necessary functions.
ShowMipIR (Minimizing Ionizazing Particle) illustrates the drift path of electrons (blue) and holes (red) in a 3D-plane. The integer inputted into ShowMipIR is ndiv, which is the number of "charge" buckets generated along the path of the ion. Think about it like the number of divisions made when approximating an integral.

The Draw function allows you to graph the electric field potential (EP), absolute electric field (EF), and the electric field along a certain axis. This is shown by specifying which plane, xy, xz, or yz, and the third variable's location.

MipIR generates the current vs. time graph by calculating the drift of both electrons and holes.

```
int ndiv = 30;

TCanvas c1;
c1.SetTitle("Illustration of Drift");
c1.cd();
det->ShowMipIR(ndiv);

TCanvas c2;
c2.SetTitle("Electric Potential X-Y
Graph");
c2.cd();
det->Draw("EPxy",99)->Draw("COLZ");

TCanvas c3;
c3.cd();
c3.SetTitle("Current vs. Time");
det.MipIR(ndiv);
det->sum.Draw();
det->neg.Draw("SAME");
det->pos.Draw("SAME");

}
```

Once you have completed all of these steps, it is possible to construct and implement an example file of your own. If you are encountering errors in the compiling and execution of your code, please consider doing the following steps. The following will outline some problems that you may encounter with your KDetSim example files.

***The example files are taking too long to run:*** The user may need to optimize the running speed of their simulation. This can be done by:

- Increasing the SetUpVolume() parameters
- Minimizing the number of divisions (ndiv)
- Minimizing the dimensions of the K3D detector function.

## KDetector Example Files

To be written soon…