OPA documentation version 4.1.0

Andreas Streun, May 16, 2025

1 How to use this document

This document is intended to support program developers, who want to maintain the existing OPA code, or to rewrite it in another environment, or to extract useful parts for integration into other codes.

When reading this document, the source code may be examined in parallel, preferably in the Lazarus IDE. In order to see how the code works, OPA may be executed or the tutorial [1] may be consulted. Further information on using OPA is found in the user guide [2] and the underlying physics is outlined in [3].

In this document following styles are used:
Pieces of Pascal code are shown as `code`. `GUI` is a graphical user interface, usually based on class `TForm`. `Frame` if a user-defined GUI-component, usually based on class `TFrame`. `Unit` is a standard Pascal unit.

A GUI or a frame has a list of `actions` corresponding to the event handlers of its components, for example a button to be pressed. The pure Pascal units are passive, they contain a set of public `procedures` which are called from the GUIs. Frames too may contain public `procedures`. Thus in this documentation, action lists are given for the GUIs and frames, and lists of public procedures for the frames and units. All modules may offer public variables, here coloured `x`, `y`, `z` for GUIs, frames, units.

GUI components accept user input and/or launch events. Some have captions, like **Buttons**, others not, like for example a (plot area) responding to mouse clicks. The general scheme is **component** → `event handler`. Event handlers may call private procedures of the GUI or public procedures from units as listed in the `uses` section. For the units `unit`>`procedure` refers to a procedure in a unit.

In this document, *red italics give informations about bugs and other problems and/or suggestions for future improvements. Green italics mention particular features which distinguish OPA from other beam dynamics and should be maintained.*

Questions may be adressed to the author at `mailto:psi@andreas-streun.de`.

2 OPA status

Following semantic versioning [4] version 4.063 of OPA was renamed 4.1.0. However, since up to now only the author worked on the code, the distinctions MAJOR, MINOR, PATCH rather refer to the changes as visible to the users.

Current version is OPA 4.1.1. It includes 44 Pascal units `.pas`, 28 of them are `GUIs` which are accompanied by a Lazarus form `.lfm` (created by the Lazarus IDE), and 16 of them are Pascal `units`. Five of the GUIs are `frames`, i.e. self-defined components embedded in other GUIs. Four units are located in a parallel folder, here called `../com/` since they are not OPA specific but used with other programs too. Table 1 lists the units. The column "Size" lists the lines of code as a rough estimate for the complexity of the unit. Furthermore, OPA includes a couple of image files `.ico, .bmp` to draw symbols on buttons.

A GUI defines a class, and the actual GUI is the one and only instance of this class, i.e. a public variable, given in table 1 in the column "self". A frame defines a class and the parent GUI will define one or more instances of this class as variables. In this case, the column "self" gives the class name.

In general GUIs should handle the user interactions only while the Pascal units do all the calculations. This separation was largely realized in OPA, however not strictly. Some minor calculations are done in the GUI in some places.

OPA was developed over more than 30 years along the design of SLS, SLS 2.0 and other machines, and features were implemented as needed for design work. The program code reflects this history and may not present the most logical structure. Also the names of the units (including inconsistent use of capitals) are historical and could be changed to more meaningful descriptors.

Like other beam dynamics programs OPA works on a lattice file. It contains *elements*, mainly magnets of different type, and *segments*, which are line-ups of elements and segments. One of the segements is selected and expanded to become *the lattice* to work with. Element parameters, e.g. quadrupole strength in the lattice file can be numbers or arithmetic expressions using *variables*, which are part of the lattice file too.

Table 1: OPA units overview

Name	self	Purpose	Size
`opamenu`	`MenuForm`	the main menu	901
`OPAglobal`		global variables and procedures	2986
`mathlib`		mathematical library	2075
`../com/Vgraph`	`Vplot`	real number graphics library	1276
`../com/asfigure`	`TFigure`	general plot procedure	314
`../com/conrect`		contour plot calculation	332
`../com/asaux`		little helpers	349
`opalatticefiles`		file reading and writing	1839
`texteditor`	`FormTxtEdt`	text editor for lattice file	190
`OPAEditor`	`FormEdit`	interactive lattice editor	346
`EdElCreate`	`EditElemCreate`	creation of an element	144
`EdElSet`		edit element parameters	861
`EdSgSet`		edit segment parameters	444
`opticview`		linear optics design	1143
`knobframe`		slider to set element property	421
`Opticstart`		setting for start parameters	615
`OpticTune`		adjustment of lattice tune	251
`OpticWOMK`		write optics markers	125
`OpticEnvel`		setting for optical functions plot	372
`OpticMatch`		linear optics matching	1457
`OpticMatchScan`		parameter scan	119
`opticplot`		linear beam dynamics calculations	2840
`OPAElements`		element propagation calculations	2387
`OPAtune`		show the tune diagram	634
`OPAmomentum`		momentum dependence and optimization	949
`MomentumLib`		momentum dependent calculations	480
`OPAChroma`		nonlinear optimization	1413
`CSexLine`		controller for nonlinear element	454
`CHamLine`		bar indicator for nonlinear Hamiltonian mode	416
`ChromGUILib1`		?	79
`ChromGUILib2`		?	93
`OPAChromaSVector`		show Hamiltonian as complex vector	145
`chromlib`		nonlinear dynamics calculations	1813
`OPAorbit`		orbit correction and injection	2353
`Bucket`		plot RF bucket	640
`LGBeditor`		longitudinal gradient bend editor	702
`LGBeditorLib`		longitudinal gradient bend calculations	471
`OPAtrackP`		phase space tracking	1422
`OPAtrackDA`		dynamic aperture tracking	1666
`OPAtrackT`		Touschek tracking	1726
`tracklib`		particle tracking calculations	1502
`OPAGeometry`		geometric machine layout and matching	2010
`OPACurrents`		export magnet currents to machine control	303
`opatest`		test of new features	472

3 Main programs and in general

3.1 Main program

`opa.lpr`

The main program file which allocates the units as described below and creates some of the forms. It is created more or less automatically by the Lazarus IDE.

3.2 Menu

`opamenu`

The main GUI for reading and writing files and to launch the other GUIs. It includes message (log) and status windows. It is always open in the background. Closing it exits OPA.

Uses: `OPAglobal`, `OPAEditor`, `texteditor`, `opalatticefiles`, `opticview`, `OPAtune`, `OPAmomentum`, `OPAChroma`, `OPAtrackP`, `OPAtrackDA`, `OPAtrackT`, `OPAorbit`, `OPAGeometry`, `LGBeditor`, `Bucket`, `OPACurrents`, `opatest`

Actions

`FormCreate` called at start creates the GUI and fills the menu items with data: the list of last used files is established and the status labels are set. Then handles to most components are passed to `OPAglobal` to make them available to all GUIs, in order to send messages to the log window, update the status in the status window and enable or disable options depending on the result of calculations. Note, that the initialization of `OPAglobal` is executed first to provide the required information.

File Menu

New → `fi_new` delete all data to start new lattice from scratch.

Open... → `fi_open` read OPA lattice file or try to read a lattice file from Tracy2, Tracy3, MAD-X, elegant or BMAD. The file name is displayed as ”active file”.

Last Used ▶ → `fi_last` select file from list of last used files

Save, Save as... → `fi_[save,svas]` save OPA lattice file with old or new name.

Export to ▶ → `ex_[tracy2,tracy,madx,elegant,bmad,opanovar]` save as file for other programs. Last option expands all arithmetic expressions in lattice and saves as OPA file.

Exit → `fi_exit` save all settings from session and exit OPA (just closing the GUI will exit OPA without saving settings).

Edit Menu

Text Editor → `ed_text` launch lattice file text editor `texteditor`.

OPA Editor → `ed_oped` launch interactive "LEGO block" editor `OPAEditor`.

Design Menu

Linear Optics → `ds_opti` launch interactive linear design `opticview`.

Off-momentum Optics → `ds_dppo` launch momentum dependent optics `OPAmomentum`

Non-linear Dynamics → `ds_sext` launch non-linear optimizer `OPAChroma`

Orbit Correction → `ds_orbc` launch orbit & injection panel `OPAorbit`

Injection Bumps → `ds_injc` launch orbit & injection panel `OPAorbit`

RF Bucket Viewer → `ds_rfbu` launch RF bucket visualization `Bucket`

Geometry Layout → `ds_geo` launch geometry layout and matching `OPAGeometry`

LGB Optimizer → `ds_lgbo` launch editor for longitudinal gradient bends `LGBeditor`

Tracking Menu

Phase Sapce → `tr_phsp` launch phase space tracking `OPAtrackP`

Dynamic Aperture → `tr_dyna` launch dynamic aperture tracking `OPAtrackDA`

Touschek Lifetime → `tr_ttau` launch Touschek lifetime tracking `OPAtrackT`

Extra Menu

Output ▶ → `tm_di` select the amount of output provided in the log window.
The implementation is incomplete and inconsistent: some output goes to the log window, some to a terminal console (only visible if it is open) and some to a file `diagopa.txt`*, which is created (but never closed...).*

Magnet Currents → `tm_cur` launch panel to calculate magnet currents `OPACurrents`

The other menu items in this group are temporary tests calling procedures from unit `opatest` and not relevant to be documented here.

(Segment name) → `ComboSeg` is a drop-down list of the available segments. The one selected is expanded to become the lattice, calling `OPAglobal`>`MakeLattice`.

Show lattice expansion → `butlatsh` displays the expanded lattice in the log window.

Print → `ButLogPrt` prints the content of the log file to `LogPrint.txt` in the working directory.

Clear → `ButLogClr` clears the log window.

3.3 Global data

`OPAglobal`

This unit defines global constants, types and variables and thus is OPA's "database". It also provides many public procedures for different non-physics tasks. Only an overview can be given here, and the most important or complex things will be explained. Everything else should become clear from the code, hopefully.

Uses: `../com/mathlib`, `../com/asaux`

Public constants, types and variables

Constants include general settings, array sizes, color definitions, name strings and values of flags later to be referenced by name.

Following type definitions may be considered as non-trivial:

`ElementType` is a case-record for defining element parameters such as length, quad focusing strength, dipole bending angle etc. Several (but not all) parameters may be arithmetic expressions instead of numbers. The "optics marker" element contains a pointer to a set of optical functions (normal mode betas and alfas, dispersions, orbit and coupling matrix) of type `OmarkType`.

`variable_type` is for variables defined by name and value or arithmetic expressions. Variables are referenced in arithmetic expressions of element parameters or in other variables.

`AbstractEleType` is for abstract elements which represent either an element or a segment and contain pointers to previous and next abstract elements. The abstract element may be inverted or repeated.

`SegmentType` is for segments, which are series of abstract elements, defined by pointers to its initial and final abstract element.

`LatticeType` is a lattice entry. The lattice is built by expaning a segment, so it contains only elements, and additional information on direction (if the element is inverted), and data relevant for orbit correction (misaligmnets and if the element sits on a girder).
Unlike several other codes OPA handles correctly nested inversions of segments to obtain the correct orientation of the element in the lattice. This is relevant for bending magnets with unequal edge properties.

`GirderType` describes a girder by first and last lattice entry of elements sitting on the girder, and by the type of connection to adjacent girders.

Other types are mainly shortcut definitions to gather interrelated data.

Only few of the global variables need an explation:

`Elem` and `Ella` are arrays of `Elementtype`, where the first one is read from file and manipulated in the editors, while the second one are a subset of elements used in the lattice and to be modified in the various calculations. After terminating a calculation the user is asked to save or cancel the changes, which causes the `Ella`-data to be copied back to the `Elem` array or not.

`Status` of `StatusType` is a group of status flags to be set by the outcome of calculations in order to enable or disable buttons and to re-use data in other calculations.

`MainButtonHandles` of `MainButtonHandlesType` provided global control of the various options in `opamenu` to enable or disable them.

`GlobDef` and `Def` of `DefaultType` are arrays of user-defined settings, which are read at start and when switching the working folder.

Public procedures

This listing of procedures is more or less inverse to the (historical and illogical) arrangement in the source file:

`Initialization` sets some global variables to reasonable initial values, sets all status flags to false, and sets default values for ∼250 user settings. Then it sets the OPA directory and tries to read the files `opa4_path.ini` which contains a list of last used files. Then it reads `opa4_glob_ini` containing global user settings (at the moment this is only the amount of output). Finally, it takes the folder from the first entry in the list of last used files to set the working folder and reads the file `opa4_set.ini` in this folder to restore the ∼250 user settings. If these files are missing, the default values are used.

`GlobDefWriteFile` and `DefWriteFile` write the user settings back to these files if the session is regularly terminated by **opamenu** → `ExitOPA`.

`MakeLattice` builds the lattice from elements and segments. Since a segment istelf contains elements and segments, the internal procedure `SegLat` is called recursively to unpack the data. The elements, which are used in the lattice are copied from the `Elem` to the `Ella` array, and correctors and monitors with generic names `CH,CV,MON` are expanded into separate instances named `CH001,CH002...` to adress them individually in orbit correction in unit `OPAorbit`. Finally status flags are set.

`GirderSetup` evaluates the girder structure if contained in the lattice and allocates the lattice elements to the corresponding girders.

`IniElem` sets default values for new elements.

`AppendAE, ClearSeg, NAESeg` are procedures to handle segments, which are series of pointers to elements or other segments.

`AppendCurve, ClearCurve` prepare data sets for plotting curves of optical functions.

`AppendChar` builds a linked list of characters for `texteditor` *(wrong place)*.

`putkval, getkval, putSexkval, getSexkval` functions are shortcuts to set or get the parameter which is considered the strength of a magnet.

`FillComboSeg` fills the list of last used files in `opamenu`.

`OPALog` writes a message to the log window in `opamenu`.

`MainButtonEnable` enables or disables the options in `opamenu` depending on the status flags. For example, tracking is only enabled if a periodic solution exists.

`PassMainButtonHandles` and `passErrLogHandle` take handles to the GUI components of `opamenu` to enable other units using `OPAglobal` to enable/disable them.

`EllaSave` and `Elcompare` check if elements in lattice (`Ella`) have been changed with regard to the original elements (`Elem`) and ask the user if the changes should be saved.

The procedures and functions listed under ”Calculator” are an arithmetic evaluator adapted from [14]. Other procedures not listed here include variable, element and segment to string conversion and other utilities which may be self-explanbatory.

The `OPAglobal` unit grew large large and heterogenous over the years and should be split into several units for the different functions to be performed. Several procedures should be moved to other units

3.4 Mathematics library

`mathlib`

A library of mathemical functions: definition of types and operations with vectors, matrices and complex numbers. It further contains some special functions, among them the Touschek integral function, and implementations of Powell's minimizer, LU decomposition and Singular Value Decomposition taken from [12], but extended to dynamic array size. OPA does not link external libraries but all algorithms needed are included in the code.

Uses: none

Public constants, types and variables

Types define vectors and matrices for beam dynamics, geometry and for the Powell minimization procedure. The only public variables are the parameters for Powell.

Public procedures

Most procedures are elementary operations on vectors, matrices and complex numbers, or for special functions, and don't need to be explained.

`CTouschek` and `CTouschek_pol` solve the Touschek lifetime integral, see appendix C.1 in [3] for details. The first procedure solves the integral based on Simpson's rule, the second procedure uses a polynomial approximation for speed-up.

`EulerAng` calculates rotation angles from a rotation matrix as explained in [5].

`LUDCMP` and `LUBKSB` peform LU-decomposition and backsubstitution to solve linear systems of equations as described in [12]. This is used in particuar for matrix inversion, `MatInv` and `MatDet`. The procedures are set fix to 5 dimensions.

`svdcmp` and `svbksb` perform singular value decomposition and backsubstitution to solver lineaer systems of equations as described in [12]. SVD is superiour to LUD for non-square and degenerate systems. An packing/unpacking algorithms was added to use the procedures with arbitrary dimension. The procedure is used in orbit correction in `OPAorbit` and for setting octupole families in `OPAChroma`.

Powell is an implementation of Powell's minimizer for steepest gradient search in N dimensions taken from [12]. It is used for sextupole optimization in OPAChroma, for longitudinal gradient optimization in LGBeditor, and (test mode only) for non-linear optimization in OPAmomentum.

3.5 Graphics library

../com/Vgraph

Class Vplot based on TObject contains a Lazarus TCanvas object. Plot commands for TCanvas use integer screen pixel coordinates. ../com/Vgraph wraps these standard plotting commands with procedures of same name but accepting physics coordinates as real numbers. Data are either scaled and forwarded to TCanvas or written to an Ecapsulated postscript file *.eps.

Furthermore procedures for crating nice axes, for drawing circles and ellipses and for grabbing the screen image have been added.

Uses: none

Public constants, types and variables

There are no public variables. A GUI will define one or more variables of class Vplot

Public procedures

Create accepts a handle to a TCanvas object, which is needed to construct a Vplot object. Further some initialization is done.

PS_start opens an *.eps file of name psfile supplied by the calling procedure for output and sets the private flag PS to true to direct the output to the file and not to the screen. A file header is written, defining the BoundingBox as the canvas screen size. Then macros for text alignment are defined in postscript language. If opening the file fails, an error message is returned.

PS_stop closes the *.eps file and sets the flag PS to false.

SetRange*, SetMargin* etc set plot ranges and scaling as displayed in Fig.1. The vertical screen coordinate counts downwards whereas the physical coordinates as well as the Postscript coordinates count upwards.

getpx,getpy and PS_getpxr,PS_getpyr translate physical coordinates x, y into screen, resp. Postscript coordinates (upper/lower line):

$$\mathtt{px} = \mathtt{px0} + \frac{\mathtt{pxrange}}{x_{\mathrm{range}}}(x - x_{\mathrm{min}}) \qquad \mathtt{py} = \left\{ \begin{array}{r} \mathtt{py0}+ \\ \mathtt{pytotal} - \mathtt{py0}- \end{array} \right\} \frac{(-\mathtt{pyrange})}{y_{\mathrm{range}}}(y - y_{\mathrm{min}})$$

AdjustAspectRatio adjusts the plot ranges such, that the unit is the same horizontal and vertical, i.e. a circle appears as circle not as an ellipse.

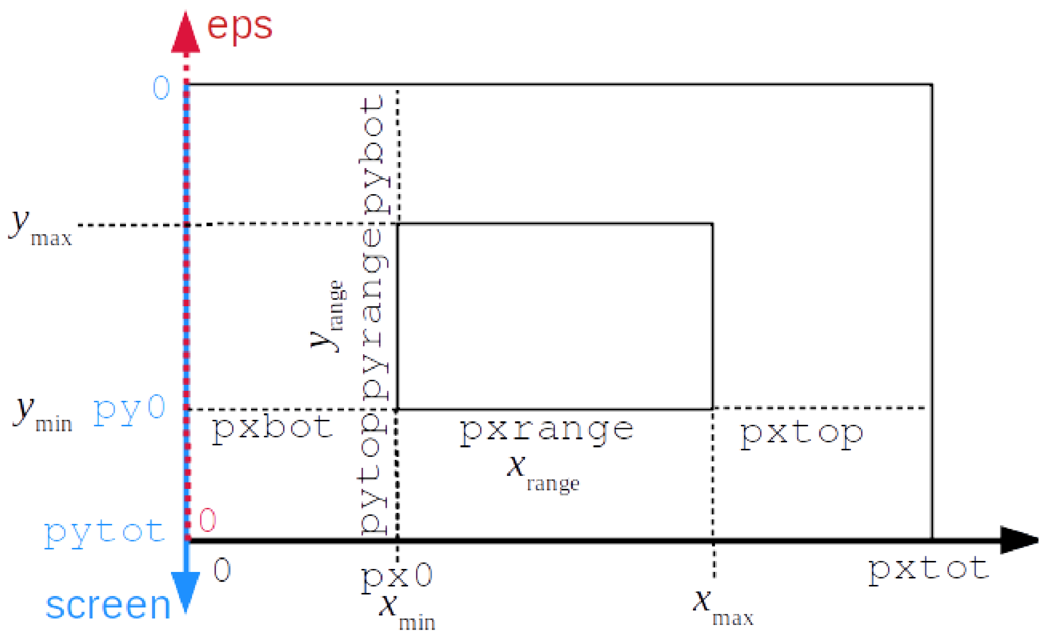

Figure 1: ../com/Vgraph coordinates

Axis plots an axis with even numbers as annotations and ranges, also extracting an exponent if needed.

GetAxisSpace returns the space required for the axis annotations without drawing the axis in order to adjust the plot range accordingely. It is called first by the calling GUI, in particular by ../com/asfigure.

Circle and Ellipse plot a circle or a (sheared) ellipse in beam dynamics notation.

GrabImage copies the screen canvas to the system clipboard, useful to catch plots for draft notes.

All the other procedures are mainly wrappers for the standard plot commands or shortcuts to draw arrows, symbols etc.

Stroke does nothing on the screen but terminates a series of plot commands in Postscript. Curve plotting procedures need to terminate a LineTo... loop by calling stroke.

Direct EPS export is a convenient function to immediately create high resolution graphics ready for publication in a LaTeXdocument.

3.6 Plot area

../com/asfigure

This frame contains a paintbox (i.e. a plot area) to be embedded in a GUI and a Vplot object using the paintbox canvas. Added functionality includes handles to edit fields in order to translate mouse actions to numbers.

Uses: `../com/Vgraph`, `../com/asaux`

Public constants, types and variables

`plot` is a variable of class `Vplot` in `../com/Vgraph`. It is accessible to the parent GUI for direct plotting.

Public procedures

`assignScreen` creates `Vplot` and passes the paintbox canvas in order to plot on the screen.

`forceMargin*` set fixed margins for the plot and supresses auto-scaling by subsequent calls to `Init` or `Vgraph`>`Axis`.

`Init` starts the plot. Inputs are the horizontal and vertical ranges, flags where to put axes and the descriptions of the axes, a flag to mark the origin and another flag to adjust the aspect ratio (i.e. increasing the smaller of the two ranges until the units are of same size, see `Vgraph`>`AdjustAspectRatio`).

`SetSize` to be called from the parent GUI on resizing is used to change the size of this frame.

`PassEditHandle*` accepts the handle to a `TEdit` field in order to link it to the mouse position inside this frame.

`PassFormHandle` accepts a handle to the parent GUI. *(?)*

`GetRange` returns the range of a rectangular region in physical coordinates, which has been marked by dragging the mouse inside the plot.

`UnfreezeEdit` frees the edit fields again, when it had been frozen by an Mouse/up event (see below).

Actions

(plot area) → `pMouseDown,pMouseMove,pMouseUp` handles mouse events inside the plot area. When edit fields are linked, the physical coordinates of the plot are displayed in the fields while moving the mouse. On releasing the mouse, the edit field freezes, i.e. saves the last data and changes its color. This is used in `OPAtrackP` to see the particle coordinates corresponding to the cursor position.
When the mouse is dragged, the region between mouse down and up is used to rescale the plot area. This is used to zoom in in `OPAGeometry` – *This works in Windows but not in Linux, probably due to different event handling – not yet understood.*

(plot area) → `pDblClick` grabs the plot image to the Clipboard on double-clicking the mouse.

3.7 Contour plot calculation

`../com/conrect`

This contour plot calculation is taken from [13] with minor adaptations. Given a 2D array of rectangular gridded data it returns an array of contour lines.

Uses: none

Public constants, types and variables

`Con...` are a set of types to store the data. `ConLinesType` contains points of a straight line belonging to an contour of height index `ih`, and `ConLinesArray` is an array of these.

Public procedures

`Conrec` is the only procedure, calculating the contours.

3.8 Utilities

`../com/asaux`

This unit is a collection of ”little helpers” for formatting numbers and strings, for defining colors etc. Some of this functions may be obsolete meanwhile, since newer versions of Lazarus and Pascal may include corresponding functions. Procedures are short and probably self-explaining. Not all of them are used in OPA (the `../com` folder units are used by several programs).

Uses: none

4 Lattice I/O and Edit

4.1 Lattice files

`opalatticefiles`

Lattice data are read from and saved to files called `*.opa`. Reading a file copies its content into a linked list of characters (basically a long string) in `opamenu`>`ReadFile`.

There is an option to read additional files of magnet calibrations and allocations (i.e. real magnet types and names associated with the element), in order to prepare data for the machine control system.

Files from other beam dynamics programs can be read and written, however, this includes only the basic parameters and may require some manual editing to make these files work. This includes Tracy-2 or -3 files `*.lat`, Elegant files `*.ele`,`*.lte`, MAD files `*.mad` and MAD sequence files `*.seq`.

Uses: OPAglobal, mathlib, ../com/asaux

Public procedures

Latread evaluates the character list of type TextBuffer: comments, enclosed by { } brackets, are skipped. An input line is terminated by a semicolon (;), its length is unlimited and it may break over several lines in the lattice file. Input lines are evaluated by searching first for a colon (:) which identfies an element or segment, or, if not found, a variable.

A variable input line contains a name, an equal sign (=) and a value or an expression. Reserved variables are found searching for the names TITLE, ALLOCATION, CALIBRATION (containing a title of the lattice and the names of optional allocation and calibration files), and by searching in the list of keywords globkeyw containing ENERGY and other global parameters. Other names not matching these keywords are considered as user-defined variables.

An element or segment input line contains a name, the colon (:) and a comma-separated list of tokens.

An element is recognized if the colon is followed by a valid element type name in list elemkeyw (which is based on ElemName in OPAglobal). Then the tokens are expected to contain each a parameter identifier, an equal sign (=) and a value or expression. There are many parameters for the different element types. The element is created and the values are assigned.

For a segment the tokens are names of elements or segments, both are assigned to a linked list of type AbstractEleType as defined in OPAglobal.

If names of allocation or calibration files were given, these files are read now.

Finally, the lattice is built from the last segment calling OPAglobal>MakeLattice.

LatReadCom is called after LatRead and searches the lattice string for a pair of tokens {com and com} to save the text between. All other comments contained in the lattice and enclosed in { } brackets only are not saved.

ReadAllocation and ReadCalibration read additional optional files containing lists of magnet types and real names and the magnet calibrations in order to calculate the magnet currents in OPACurrents. An example for SLS is found at https://ados.web.psi.ch/slsdesc/optic/magnets.html.

The ElementType in OPAglobal contains a pointer to a linked list of NameListType, which contains real names of elements, name of power supply, i.e. control channel, polarity etc.

lteconvert, madconvert and madseqconvert try to make Elegant *.lte and MAD *.mad and *.seq files understandable for OPA. They work on the TextBuffer string by exchanging keywords and inserting conversion factors without analyzing the lattice itself. This is then done by a subsequent call to LatRead. Tracy *.lat and Elegant *.ele files can be digested directly by LatRead.

`WriteLattice` accepts a variable `mode` of value `0..5` to write an OPA, Tracy-2, Elegant, MAD-X, BMAD or Tracy-3 file. `mode` `6` writes an OPA-file with all arithmetic expressions expanded into numbers.

Different beam dynamics codes use different units (e.g. degrees or radians) and different definitions of magnet order and strength. Only OPA, Elegant and Tracy-3 handle correctly nested inversions of segments to implement an asymmetric element (e.g. dipole with different entry and exit edge angles) correctly in the lattice. Export to the other programs requires to create and insert inverted segments (prefix `I_`) for all segments containing asymmetric elements.

All programs mentioned above allow arithmetic expressions instead of plain values for element parameters. However, unlike the others, Elegant uses inverse Polish notation, which is not included here, therefore expressions are expanded, i.e. resolved to values, for Elegant.

Further export of data is straightforward: first variables, then elements, then segments. The complete lattice file is returned as one long string.

Type `TextBuffer` is a linked list of characters and has historical origin. It could be replaced by a long string, which would be much simpler.

4.2 Text Editor

`texteditor`

A simple Notepad-like text editor to edit the lattice file. The content of the edit window is one variable of type `textbuffer`, i.e. the editor "does not know" what the content means. At exit or when a test is requested, `opalatticefiles`>`LatRead` is used to analyze the lattice.

Uses: `OPAglobal`, `opalatticefiles`

Public procedures

`Init` called from `opamenu` accepts a handle to the drop down list of segments in order to change its content when segments are created or deleted.

`LoadLattice` called from `opamenu` calls `opalatticefiles`>`WriteLattice` to write the lattice file into the text window `EdtWin`. Then the text window is copied into a text buffer to save its content.

Actions

(text window) → `disableOKBut` disables the **OK** button when text is changed. No other action is done on text input.

Test → `filetest` performs a test of the input by calling `opalatticefiles`>`LatRead`. Before, the input is copied into a buffer, then, by calling `OPAglobal`>`PassErrLogHandle`, the output from `LatRead` is redirected from the `opamenu` main log window to the local error log window `myerrlog`, because the main GUI may be covered by the text editor GUI. If the test is successful, the **OK** button is enabled. Then the handle for output is set back to the main GUI.

OK → `fileget` reads the lattice in the same way, now knowing that the input is a valid lattice, and exits. At Exit, the segment drop down list in the mnain GUI is set to the last segment, from which the lattice was built by `opalatticefiles`>`LatRead`.

Cancel → `fileRestore` ignores the text window, reads the lattice from the buffer as saved at `Init` and exits.

Nowadays better text editors may be available, which include colored mark-up, auto-complete options and real time checks.

4.3 Interactive Editor

`OPAEditor`

The "OPA Editor" allows to create elements and compose a lattice without knowing element type names and lattice file syntax. It is more suitable for beginners, while the text editor may be more convenient for experienced users.

The GUI displays two lists, one for variables and elements, the other one for segments. In addition some global parameters like beam energy and default magnet aperture may be set.

Uses: `OPAglobal`, `EdElCreate`, `EdElSet`, `EdSgSet`, `../com/asaux`

Public procedures

`Init` called from `opamenu` accepts a handle to the drop down list of segments in order to change its content when segments are created or deleted. Init is also called internally to update the list contents.

Actions

(Element list) → `ListBoxEClick`: if the first line of the list is licked, the object `EditElemCreate` of class `EdElCreate` is initialized by passing handles to this list and to the `EdSgSet` object for editing segments. If another line of the list is clicked then object `EditElemSet` of class `EdElSet` is intitalized by calling two different procedures depending on if the clicked item represents a variable or an element, and a handle to this list is passed.

(Segment List) → `ListBoxSClick` initializes object `EditSegSet` of class `EdSgSet` by passing a handle to this list.

Beam energy → `EditGloEKeyPress`, (aperture fields) → `EditA*KeyPress`,
Magnet pole radius → `EdrrefKeyPress` are edit fields to accept values for beam energy, for element default apertures, and for default magnet pole inscribed radius.

(Comment window) (no event) text window `MemCom` accepts a comment, which will be saved in the lattice file between the {`com...com`} tokens.

Invert all dipole polarities → `ButDipInv` inverts all dipole polarities.

Expand undulators → `ButExpUndClick` expands an undulator into a series of dipoles and drift spaces creating the elements and re-defining the undulator as segment. Usually undulator is a basic element, and the series of dipoles and drifts is executed internally and not visible to the user.

Set all apertures → `ButAllAperClick` sets all apertures to the values given in the `EditAx`, `EditAy` fields thus overwriting individual element settings.

Set all (not only if larger) (no event) if checkbox `ChkAll` is checked, apertures of all elements are set, otherwise only those which have a larger aperture than the values in the fields.

Invert rotations in inverted segments (no event): If checkbox `ChkGloRi` is ticked or not defines if a beam rotation is inverted (like an asymmetric elements) or not when a segment containing the rotations is inverted. A corresponding flag `glob.rot_inv` is set and saved in the lattice file.

Exit → `ButExit` saves data, updates the segment drop down list in `opamenu` and closes the GUI.

4.4 Element Creator

`EdElCreate`

This small GUI pops up, when `new entry` is clicked in the `OPAEditor` elements and variables list. A new element or variable is created after selecting its type and giving it a name. Then the GUI is closed and the `EdElSet` GUI pops up.

Uses: `OPAglobal`, `EdElSet`, `EdSgSet`

Public procedures

`Init` accepts handles to the `OPAEditor` elements list and to the segment editor `EdSgSet`, because if an element is created or subsequently modified by `EdElSet`, the changes should appear in these two GUIs. Then the list of element types is filled with the types defined in `OPAglobal`.

Actions

(Kind) → `ComETypeChange` selects the type of element from the list.

(Name) → `EdENameChange` accepts the name of the element.

Create → `ButCreClick` tests the input, if a type was defined, if the new entry is a variable or an element, and adds it to the corresponding arrays. Then the GUI closes itself and launches `EdElSet` using one of the two initialization procedures for variables and elements. If it is an element, in case the segment editor `EdSgSet` is open, it is updated.

Cancel → `ButCanClick` closes the GUI doing nothing.

Bug: there is no check for duplicate entries, should be added.

4.5 Element Editor

`EdElSet`

Table to edit all parameters of an element or a variable. Some fields may contain algebraic expressions to calculate parameters from variables. This procedure is called in four different ways, for elements and for variables, and from editor or from optics design. For (iron dominated) magnets, a possible pole-profile is plotted `>PlotPro`. **Uses:** `OPAglobal`, `opticplot`, `../com/asfigure`, `../com/asaux`

4.6 Segment Editor

`EdSgSet`

A table to set up or modify a segment, which is a line-up of elements and segments. **Uses:** `OPAglobal`, `../com/asaux`

5 Linear Optics

5.1 Linear optics design

`opticview`

5.2 Parameter knob

`knobframe`

5.3 Start parameters

`Opticstart`

5.4 Tune matching

`OpticTune`

5.5 Optics marker update

`OpticWOMK`

5.6 Plot mode selection

`OpticEnvel`

5.7 Linar matching

`OpticMatch`

5.8 Parameter scan

`OpticMatchScan`

5.9 Lattice calculations

`opticplot`

5.10 Element calculations

`OPAElements`

A beam as defined by its orbit, (normal mode) beta functions, dispersion and coupling matrix is propagated through an element `>Driftspace,Quadrupole...`

Internally the procedures `>MCC_prop` and `>MBD_prop` perform the transfer for coupling and non-coupling eleements.

Depending on flags set, Kickers, non-linearities, misalignments etc. are switched on or off, and radiation integrals, path length etc. are calculated.

For display OPA subdivides elements in slices, where the slice length corresponds to one pixel on the screen. This is handled by the procedures `>slice...` defining matrices for the slices and for the skipped parts (i.e. out of view range) of the element.

Internal private procedures calculate phase advances and transform between physical, normal modes and normalized normal mode coordinates.

5.11 Tune diagram

`OPAtune`

6 Momentum Dependent Optics

6.1 Momentum dependence

`OPAmomentum`

6.2 Calculations

`MomentumLib`

7 Non-linear Optimization

7.1 Non-linear optimization

`OPAChroma`

7.2 Element controller

`CSexLine`

7.3 Result indicator

`CHamLine`

7.4 Result handles

`ChromGUILib1`

7.5 Element handles

`ChromGUILib2`

7.6 Vector diagram

`OPAChromaSVector`

7.7 Calculations

`chromlib`

8 Orbit and Injection

8.1 Orbit and injection

`OPAorbit`

9 Special Element Editors

9.1 RF bucket

`Bucket`

9.2 Longitudinal gradient bend

`LGBeditor`

9.3 LGB calculations

`LGBeditorLib`

10 Tracking

10.1 Phase space

`OPAtrackP`

10.2 Dynamic aperture

`OPAtrackDA`

10.3 Touschek lifetime

`OPAtrackT`

10.4 Calculations

`tracklib`

11 Layout, Currents etc

11.1 Machine Layout

`OPAGeometry`

11.2 Magnet currents

`OPACurrents`

`getKfromI, getdKdIfac, getIfromK` convert magnet strength in current and vice versa taking into account non-linear magnet saturation.

11.3 Test procedures

`opatest`

11.4 Editors

`LGBeditor.pas/.lfm` (19.1k)

---

uses `OPAglobal, MathLib, ASfigure, ASaux`

`LGBeditorLib.pas` (13.9k)

---

uses `OPAglobal, ASfigure, ASaux`

These two procedures optimize the field profile of a longitudinal gradient bend in order to minimize quantum excitation, i.e. the I_5 radiation integral. Magnet type, length, peak field and number of slices are set before starting a Powell minimizer. The first procedure is mainly the GUI and the minimizer, the second one contains physics and plotting.

11.5 Optics Design

`OpticView.pas/.lfm` (32.3k)

---

uses `OPAglobal, OpticPlot, Opticstart, OpticEnvel, OpticTune, OpticMatch, OpticWOMK, OPAtune, knobframe, EdElSet, Mathlib, VGraph, ASaux.`

This is the main GUI for (linear) optics development. It contains procedures for initialization the form, for plotting, handlers for mouse events and GUI buttons.

`OpticPlot.pas` (98.3k)

---

uses `OPAglobal, OPAElements, OPAtune, MathLib, ASfigure, VGraph, ASaux.`

Linear beam optics includes closed orbit finder `>CloseOrbit`, periodic solution for coupled and uncoupled lattices `>NormalMode, FlatPeriodic`, propagation through elements `>Lattel` and full linear optics including integrals `>LinOp` with several options for periodic, symmetric, single pass forward, backward, from marker calculations `>OpticCalc`. Plot routines show beta functions and dispersions `>PlotBeta`, or orbit, envelopes and apertures `>PlotOrbit, PlotEnv, PlotApertures`, or magnetic fields `>PlotMag`. Lattice parameters are shown in a table `>FillBeamTab, FillBetaTab` and several output files can be printed `>Print...`

`Opticstart.pas/.lfm` (17.3k)

uses `OPAglobal, OpticPlot, MathLib, ASaux.`

GUI to select the starting conditions for the optics solution, either periodic/symmetric, or from initial values for orbit, beta functions, dispersion and coupling given at the start or end of lattice or at an intermediate optics marker.

`OpticEnvel.pas/.lfm` (9.9k)

uses `OPAglobal, OpticPlot, ASaux.`

Select the plot mode to show beta functions (normal mode and/or projected) and dispersions, envelopes with orbit and apertures or magnetic fields. Several parameters may be set, e.g. the reference radius to calculate magnetic fields etc.

`OpticTune.pas/.lfm` (6.1k)

uses `OPAglobal, OpticPlot, MathLib, ASaux.`

Calculates a 2×2 matrix how the tunes depend on a scaling factor applied to all focusing and defocusing quads and allows the working point to shifted this way.

`OpticWOMK.pas/.lfm` (2.5k)

uses `OPAglobal, OpticPlot.`

Asks whether an optics solution should be stored in the optics markers.

`OpticMatch.pas/.lfm` (42.0k)

uses `OPAglobal, OpticPlot, OpticMatchScan, Knobframe, ASfigure, ASaux.`

Matching of linear optics using a Newton-Raphson minimizer: Optics parameters to be adjusted with target values and at least the same number of knobs (e.g. quadrupole strenght, variables etc.) are selected, then the minimizer proceeds using the square matrix of most sensitive knobs, which is recalculated after each steps to take into account non-linearities. Reduced step size can be set for better convergence. It is possible to run a scan over some range of a target value. *The matching procedure is quit old and restricted to uncoupled lattices. A more elaborate minimizer also including limits for the knobs should be implemented.*

`OpticMatchScan.pas/.lfm` (2.9k)

uses `OPAglobal, OpticPlot, ASaux.`

This is just a GUI to enter the range for a target function to run a scan.

`knobframe.pas` (11.5k) [Frame]

---

uses `OPAglobal, OpticPlot, Mathlib, ASaux.`

The frame provides a knob to control a parameter (quad strength, variable value etc.) including limits and a reset function. A numner of knob frames is embedded in the GUI `OpticView, OPAOrbit` as space permits. Knob actions trigger calculations and plots. Knobs may be passive, if the quantity is controlled by a variable connected to another knob. Therefore a knob has to know his fellow knobs and trigger their updates `>BrotherHandles`

`OPAtune.pas/.lfm` (15.9k)

---

uses `OPAglobal, ASfigure, ASaux.`

GUI window to show a tune diagram: resonances up to selected orders and the working point, a line or a group of tune points is shown in the diagram. This procedure is used in linear optics to show the working point or it variation with momentum, in non-lineare optics to show a predictions for the tune footprint and in tracking to show the tracked tune footprint.

`OPAorbit.pas/.lfm` (73.0k)

---

uses `OPAglobal, OpticPlot, OpticStart, Knobframe, MathLib, ASfigure, ASaux.`

This unit calculates the beam orbit. It is used to either study orbit distortion and correction (usually in periodic mode) or to study injection (usually in forward mode) – a corresponding flag is set at start. Most procedureds in the unit are used to set up the rather complex GUI with knobs for correctors, kickers and BPMs and various panels for orbit correction, plotting or injection studies.

In orbit mode, correlated misalignments are applied, the response matrix is calculated and pseudo-inverted using an SVD procedure in order to correct the orbit. A loop function may run several error seeds to obtain some statistics.

In injection mode, kickers may be synchronized for correct timing, and the injected (and/or stored) beam trajectory is calculated for a few turns.

In both modes, when leaving the unit, results for misalignments and corrector settings, or for kicker settings, are saved internally and will be used in subsequent linear optics calculations or tracking.

There are problems with misalignments in tracking, not yet solved.

Note, that correctors and BPMs defined with reserved names `CH,CV,MON` in the lattice file are internally expanded to a set of individual elements, e.g. `CH001...` when unpacking the lattice `OPAGlobal>MakeLattice`.

11.6 Longitudinal optics

OPA does not contain true longitudinal dynamics, i.e. all calculations are for fixed momentum offset, and tracking proceeds only in 4-D, not in 6-D. There are no cavities and no acceleration.

`OPAmomentum.pas/.lfm` (26.8k)

---

uses `OPAglobal, OpticPlot, ASfigure, OPAtune, MathLib, ASaux.`

`MomentumLib.pas` (16.5k)

---

uses `OPAglobal, OpticPlot, ASfigure, OPAtune, MathLib, ASaux.`

These two procedures calculate linear optics (periodic or forward) for some momentum range, show the results and apply a polynomial fit. The first procedure mainly controls the GUI, the second one is for calculation and plotting. Results to fit and show are selected in the GUI. Results for path length are saved internally to be used when plotting the bucket (see next unit). Tune results are shown in a tune diagram.

A Powell minimizer was implemented once for a special purpose (non-linear bunch compressor study): it takes selected multipoles as knobs to adjust a momentum dependent function (e.g. $X(\delta)$) to a target function. However this implementation was "quick and dirty" and may be removed again.

`Bucket.pas/.lfm` (19.4k)

---

uses `OPAglobal, ASfigure, MathLib, Conrect, ASaux.`

The RF bucket is calculated based on up to five orders of momentum compaction and RF harmonics as described in Appendix **??**, `>CalcBucket`, and a contour plot of equipotentials and the separatrix is shown.

The coefficients for momentum compaction may be taken over from the previous unit on momentum dependent optics. The estimates for momentum acceptance and bunch length are saved for re-use in the Touschek tracking unit (see below).

11.7 Non-linear Optimization

Non-linear optimization uses a penalty function composed from a fixed set of Hamiltonian modes on one side, which are controlled by a variable set of sextupoles and octupoles on the other side. The dependencies are complicated, if the two minimizers are running (Powell for first and second order sextupole terms, SVD for first order octupole terms), or if chromaticity is automatically adjusted, or if target values for the (non-resonant) Hamiltonian modes are changed. This requires to pass handles from the GUI program `OPAChroma` to the frames for the Hamiltonian modes and the multipoles, `>CSexLine, CHamLIne` and between these. In order to avoid circular dependencies, two intermediate layers `>ChromGUILib1,2` were introduced.

The nested handles are rather messy. It works, but could be entangled and simplified.

`OPAChroma.pas/.lfm` (40.2k)

---

uses `OPAglobal, ChromLib, ChromGUILib1, ChromGUILib2, CHamLine, CSexLine, OPAChromaSVector, OPAtune, MathLib, ASfigure, ASaux.`

Initialization of the GUI, allocating the frames for Hamiltonians and multipoles and passing all the handles between, `>Start` and dimensioning of the GUI `>ResizeAll`. Set up (and start) the Powell minimizer for the sextupoles `>ButMinClick` and set up the SVD-minimizer for the octupoles `>ChkOctClick`.

`ChromLib.pas` (60.2k)

---

uses `OPAglobal, OpticPlot, OPAElements, OPAtune, MathLib, Vgraph, ASaux.`

The physics part: at start, all kicks from sextupole, octupole, combined function bend (and decapole) families are collected, and matrices are set up to get the Hamiltonian modes from the $M_{\rm sx}$ sextupole and $M_{\rm oc}$ octupole families. These are a $10 \times M_{\rm sx}$ matrix for first order sextupole, a $11 \times M_{\rm sx}^2$ matrix for the second order sextupole, and a $13 \times M_{\rm oc}$ for first order octupole. Matrix elements are sums over optical functions at all kicks (thick sextupoles contain several kicks). Quadrupoles contribute to chromatic modes and are included as a constant offset vector. These calculations are rather time consuming but only done once at start `>ChromInit, S_Matrix`. Hamiltonian modes are calculated for the lattice structure considered as one period. In order to get the results for many periods, complex multiplication factors are required as described in Appendix **??** `>S_Period`.

Chromaticity up to third order is calculated from numerical differentiation `>ChromDiff`. Hamiltonian modes are calculated from multipole settings using the pre-calculated matrices and weight factors entered manually in order to visualize the results and combine them into a single, scalar penalty function, see Eq.**??** `>Driveterms`. Chromaticity is corrected with two selected sextupole families using a simple 2×2 matrix `>UpdateChromMatrix, GetLinChroma, ChromCorrect`.

For the octupole matrix a SVD decomposition is performed for correction of the second order sextupole terms which are first order octupole terms. The SVD weight vector is provided for filtering, since usually a “hard” correction using all weights does not work well `>Oct_SVDCMP`. Decapoles affect only the third order chromaticities. Other third order effects are not included.

`ChromGUILib1.pas` (1.9k)

---

uses `ChromLib, CHamLine.`

This small piece of code is mainly for passing handles to the`>CHamLine` frames.

`ChromGUILib2.pas` (2.7k)

uses `ChromLib`

Pass handles to the octupole SVD functions in order to enable automatic execution during minimization, and filters the SVD weight factors.

`CHamLine.pas/.lfm` (10.7k) [Frame]

uses `OPAglobal, ChromLib, ChromGUILib2, CSexLine, ASaux.`

25 of these frames are embedded in the GUI at start. They display the results for the 25 Hamiltionian modes and allow weight and target values to be set. A change of target or weight triggers an update of the calculations `>UpdateWeight, UpdateTarget`. Changing any multipole manually or by the minimizer of course changes these frames.

`CSexLine.pas/.lfm` (11.9k) [Frame]

uses `OPAglobal, ChromLib, ChromGUILib1, ChromGUILib2, OPAChromaSVector, ASaux.`

At start, one of these frames is embedded in the GUI for each family of sextupole, octupole, decapole and combined function bend. It provides a knob for the magnet strength. Changing it triggers a calculation of the Hamiltonian modes. If the minimizer is running, the strength field has to follow `>UpdateVal...`

`OPAChromaSVector.pas/.lfm` (3.3k)

uses `OPAglobal, ChromLib, MathLib, ASfigure.`

Opens a small window to visualize the first order sextupole modes in the complex plane.

11.8 Tracking

Tracking is performed in unit `TrackLib`, used by the three GUIs for phase space, dynamic aperture and Touschek tracking:
`OPAtrackP.pas/.lfm` (44.8k)

uses `OPAglobal, OPAtune, TrackLib, MathLib, Vgraph, ASfigure, ASaux.`

Phase space tracking: the GUI contains four plot windows for the transverse phase spaces and Fourier spectra, and several panels for different options. Single particles are started from coordinates entered manually in edit fields or passed from `ASfigure` mouse events `>ButRunClick`. For amplitude dependent tune shifts (ADTS) particle start coordinates are stepped up to aperture limits `>ButTushClick`. For simulation of injection, a beam ellipse populated with many particles can be tracked `>StartEnsemble, ButBeamRunClick, TrackBeam`. Most of the unit contains event handlers and plot routines.

`OPAtrackDA.pas/.lfm` (30.9k)

---

uses `OPAglobal, OPAtune, TrackLib, MathLib, Vgraph, ASfigure, ASaux.`

Dynamic aperture tracking: particles are started on a grid covering the area (x, y), (x, δ) or (y, δ). The grid is successively refined to get an early impression `>gridSetup`. Physical apertures are calculate by projecting all apertures to the trackpoint `>Silhouette`. The other procedures are for event handling and plotting.

`OPAtrackT.pas/.lfm` (45.7k)

---

uses `OPAglobal, OPAtune, TrackLib, OpticPlot, MathLib, Vgraph, ASfigure,ASaux.`

Touschek tracking: Lifetime is calculated as described in Appendix ?? from bunch volume and momentum acceptance (MA). The MA is the minimum of the linear MA given by the beam pipe apertures `>CalcMALin`, of the RF-MA which is derived from input parameters or has been calculated previously by `Bucket`, and of the dynamic MA obtained from tracking and binary search for min./max. stable momentum offset `>MADyn, Trackdbins, TrackLat`.

The bunch volume is calculated from the periodic optics solution with its emittance and energy spread `>CalcSigma`.

The GUI has two panels for several input values affecting lifetime and for derived values `>Output`.

Coulomb lifetime is calculated from the effective acceptance, hower this includes only the physical, not the dynamic aperture limits `>CalcAccEL`. Bremsstrahlung lifetime uses the negative MA as used for Touschek lifetime too. Total lifetime is given as the inverse of the sum of the loss rates `>CalcLifetime`.

`TrackLib.pas` (46.2k)

---

uses `OPAglobal, OpticPlot, OPAElements, MathLib;`

At start, all linear elements are concatenated to matrices alternating with non-linear (or time dependent) kicks in order to speed up tracking `>TrackinMatrix`. Physical acceptances are calculated either from element apertures or from given apertures to estimate the maximum range for tracking `>Acceptances`. Both calculations have to be re-done when changing the reference momentum `>Init_dpp`. The available aperture in the (x, y)-plane is calculated for a set of rays to obtain the silhouette, i.e. projection of beam pipe apertures to the trackpoint as described in Appendix ?? `>AmpKappa`. Optics parameters at the Trackpoint are calculated, which may be located anywhere, even inside an element `>TrackPoint`.

Tracking proceeds turn by turn at fixed momentum `>OneTurn` (or with changing momentum based on a simple model of synchtron oscillation `>OneTurn_S`). Tracking one turn proceeds by repeated application of the matrix for a series of linear elements, followed by a non-linear (or time dependent) kick and a check for particle loss `>TMatKick`.

The procedures in the last third of the unit (from `SineWindow` to the end) are for signal processing, to calculate the FFT, interpolate frequencies and guess the related resonances.

11.9 Lattice Layout

`OPAGeometry.pas/.lfm` (63.2k)

---

uses `OPAglobal, OpticPlot, MathLib, ASaux.`

This unit displays the lattice layout, peforms geometric matching and exports various files. The orbit is calculated in 3D-space from element lengths, deflection and rotation angles `>CalcOrbit`. The orbit as curve in space is rotated and translated depending on the initial conditions `>setDrawMode`. The elements are made from faces, i.e. polygons in 3D-space `>CalcFaces`, which become polygons when projected to 2D-space of the image plane `>CalcPoly`. Changing the initial conditions does not change the faces themselves but applies the same translationa and rotation to all of them `>TransPoly`.

Up to now the elements are "flat", i.e. represented by a face of some length and width in the midplane. If changing the initial angle, they look ugly. It would be straightforward to define boxes made from several faces, however this would also require some rendering algorithm for 3D-display.

Several files can be exported, among them a `.geo` file of polygons `>butListClick`, which can also be read `>ReadFiles` in order to show several lattice structures in one plot.

Geometric matching looks for lattice variables with names starting with "G", which control lengths and deflection or rotation angles of elements (in `>Start`) and runs an SVD minimizer to adjust the final (or initial) coordinates and angles of the lattice structure to a target value `>butMatchClick`.

Geometric matching works but is not well implemented yet. More checks are required, and also an "undo" button should be added.

11.10 Temporary

`OPACurrents.pas/.lfm` (10.3k)

---

uses `OPAglobal, ASaux.`

This unit calculates magnet currents using two tables for allocation (i.e. hardware type of the magnet in the lattice) and magnet calibration, which may be read with the lattice in `OPAGlobal> ReadAllocation, ReadCalibration`. Calculating the current from the strength parameters and v.v. is done in `OPAGlobal> getIfromK, getKfromI`. Due to non-linearity of the calibration curve, a bisection root finder is used [12].

The `OPACurrents` unit provides a GUI and writes a `.snap` file to be read by the storage ring control system. Among other data, this file also includes the matrices for tune change, if previously calculated by `OpticTune`, and for chromaticity change, calculated by `ChromLib>UpdateChromMatrix`.

The current calculation should be done here, not in `OPAGlobal`*.*

`opatest.pas` (37.3k)

---

uses `OPAglobal, OpticPlot, MathLib, ASaux.`

This unit is for temporarily needed procedures or for testing new stuff, which then, if it works well, is moved to another place.

In the present (May 16, 2025) version the unit contains, among others, three procedures for reading MAD `.mad`, MAD sequence `.seq` and ELEGANT `.lte` files. The procedures are yet incomplete but nevertheless facilitate reading these files.

References

[1] OPA Tutorial, `https://andreas-streun.de/opa/tutorial2.pdf`

[2] OPA userguide, `https://andreas-streun.de/opa/opa4.pdf`

[3] Inside OPA, `https://andreas-streun.de/opa/inside.pdf`

[4] Semantic versioning, `https://semver.org/`

[5] G. Slabaugh, Computing Euler angles from a rotation matrix, `https://eecs.qmul.ac.uk/~gslabaugh/publications/euler.pdf`.

[6] V. Ziemann and A. Streun, Equilibrium parameters in coupled storage ring lattices and practical applications, Phys. Rev. Accel. Beams, 25, 050703, 2022. `https://link.aps.org/doi/10.1103/PhysRevAccelBeams.25.050703`.

[7] D. Edward and L.Teng, *Parametrization of linear coupled motion in periodic systems,* IEEE Trans.Nucl.Sci. 20, 885 (1973).

[8] D. Sagan, D. Rubin, *Linear Analysis of coupled lattices,* Physical Review Special Topics–Accelerators and Beams 2 (1999) 074001.

[9] J. Bengtsson, The sextupole scheme for the SLS, SLS-Note 9/97, `http://slsbd.psi.ch/pub/slsnotes/sls0997.pdf`

[10] Chun-xi Wang, Explicit formulas for 2nd-order driving terms due to sextupoles and chromatic effects of quadrupoles, ANL/APS/LS-330, 2012

[11] S. C. Leemann and A. Streun, Perspectives for future light source lattices incorporating yet uncommon magnets, Phys. ST Rev. Accel. Beams, 14, 030701, 2011.

[12] W. H. Press et al., Numerical Recipes in Pascal, Cambridge 1989.

[13] `http://paulbourke.net/papers/conrec/`

[14] `http://rosettacode.org/wiki/Arithmetic_Evaluator/Pascal`