Beam Physics Note
TRI-BN-20-01R
February 24, 2020/R2020-06-08

Longitudinal Beam Topology with TRANSOPTR

Olivier Shelbaya

TRIUMF

Abstract: This note details two python scripts which have been written, enabling the sequential running of TRANSOPTR in element configuration space scans. This is of particular interest for the study and characterization of the longitudinal beam dynamics of linear accelerating cavities as are operated at ISAC. The first script, topology.py, enables 2-parameter scans of TRANSOPTR on a user specified square grid in configuration space. The second, pathFinder.py, reads-in the first's output and performs peak analysis allowing the extraction of optimum parameter pairs. A brief use example is provided in the case of the 11MHz HEBT buncher, whose longitudinal output is analyzed.

1 Introduction

The library `acc`[1] enables the running of `TRANSOPTR` in the `python` programming language. This, coupled with the envelope code's fast execution time, generally well below one second for most paths at TRIUMF-ISAC, opens the possibility of surveying the parameter space of elements in the code.

Of central interest for linear accelerating cavity tuning is the determination of the voltage-phase output longitudinal beam properties. In other words, for a given combination of (V_s,ϕ) in a cavity, what are the resulting output centroid energy and longitudinal bunch dimensions at a given target point downstream?

In this note, the novel use of the code `TRANSOPTR`, performing topological surveys on the landscape of output accelerating cavity longitudinal dynamics, is reported. As these surveys represent the output beam properties as a function of both operationally available tuning parameters for linear accelerating cavities, they are of high relevance to the operation of such devices.

The package `xml2optr` is used to produce arbitrary beamline segments using sequences of elements defined in the HLA `acc` database. For the present report, the ISAC-HEBT beamline `/acc` database implementation was used. For the purposes of studying the longitudinal dynamics of the HEBT 11MHz buncher, a segment was defined beginning at the output of the ISAC-DTL and terminating at the location of the DRAGON gas chamber. This is shown in Figure 1. As the process of generating the `TRANSOPTR` files is entirely automated, this combined with the `python-acclib` means the simulations can be run as part of `python` scripts which may include loops, optimizations, etc..

[1] Credit: Paul Jung, TRIUMF/University of Waterloo and Thomas Planche, TRIUMF.

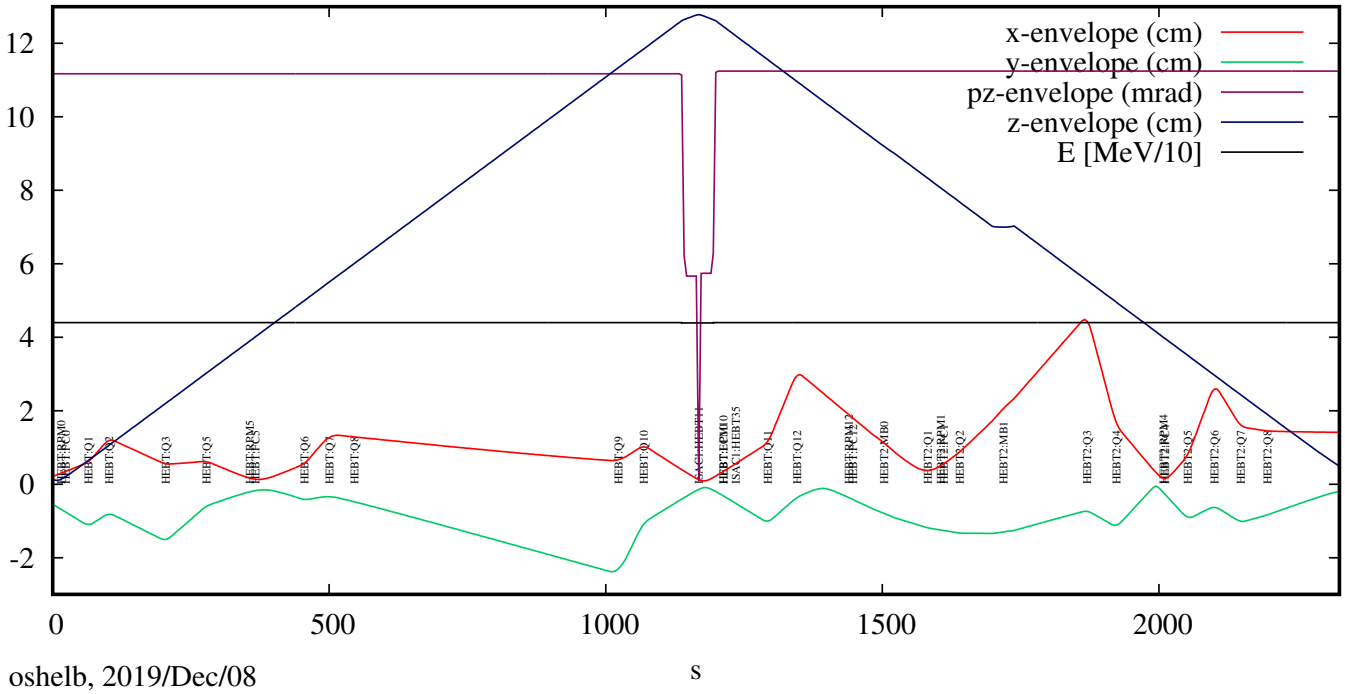

Figure 1: TRANSOPTR simulation of the HEBT beamline, showing $^{22}Ne^{4+}$ at 200 keV/u, from the DTL output up to the DRAGON experiment, as generated for and used in this work. The longitudinal coordinate s is in centimeters. The acc database was used to generate this simulation [1].

2 topology.py

topology.py [2] was originally written for the purposes of studying the longitudinal dynamics of the ISAC linear accelerating cavities. However, the script will work with any two desired tuneable parameter in a TRANSOPTR simulation. The /acc database path ios-mws-hebt2-dragon and a tune defined as design_hebt-dragon.xml, both of which are listed in the appendix, are used to build sy.f. The path itself calls sequences defined after the ISAC-DTL, which are detailed in [1].

topology builds the TRANSOPTR sequence with the supplied tune and beam parameters: 200 keV/u $^{22}Ne^{4+}$, which would correspond to tank-1 DTL output. The starting parameters have been taken from original LANA simulations of the DTL, detailed in [3]. The script iterates the phase and voltage parameters in data.dat at each run. The iterations are carried out in an N by N grid, specified by the user.

The files associated with the simulation are only built once, and the original `data.dat` file is copied to `data.backup` for later reference. As the script iterates through the (V_S, ϕ) space of the cavity, running the `TRANSOPTR` sequence back to back for each of the iterated parameters. At each iteration, the output `TRANSOPTR` data is stored in an array, allowing for them to be saved to memory. In the case of the longitudinal beam dynamics investigations for ISAC, coordinates 5 and 6 are stored:

$$z = \beta c \Delta t \tag{1}$$

$$P_z = \frac{\Delta E}{\beta c} \tag{2}$$

A standard file header is also printed for reference, consisting of a few lines of information. Importantly, the second line contains only an integer, which documents the remaining number of header lines, for later post-processing with separate tools. An example header is shown below, for an 11 MHz HEBT buncher scan, on a 75x75 grid:

```
TRANSOPTR topology scan
3
ISAC1:HEBT11
grid dimension, min phase (deg), max phase, min volt (V), max volt
75 0.0 360.0 0.0 2000000.0
```

In the above, the buncher was scanned from 0° to 360° and from 0 to 2.0 MV, the operational limit for the device taken from [4]. The next line in the output file would consist of 75 sequential entries with output `TRANSOPTR` data, corresponding to the user-specified output for each step in the iterations. Of note is that the scans consist of two nested `for` loops, with the outer loop iterating through phase and the inner loop through voltage. This means that each line of the output file is for a fixed phase, with each entry being gradually increasing voltage.

For clarity, the phase and voltage loops are shown below:

```
tankName="ISAC1:HEBT11"
phasePv=tankName+":PHASE:SETPT"
ampPv=tankName+":AMP:SETPT"
grid=75

for j in range(0,grid):


    deltaPhi = (phaseMax - phaseMin)/(grid-1)
    mod_pv[phasePv] = float(j)*deltaPhi
```

```
for i in range(0,grid):

    ampValue = (voltMax - voltMin)/grid
    mod_pv[ampPv] = float(i)*ampValue

    envelope = run_optr(output_dir,mod_pv)
    c5_out+=str(envelope[-1][5])+"  "
    c6_out+=str(envelope[-1][6])+"  "

    datadat = DataDat(file=output_dir+"data.backup")

    #compute initial beta, gamma (relativistic)
    gami = datadat['energy']/datadat['mass'] + 1.0
    beti = math.sqrt(1 - 1/float(gami)**2)

    #compute final beta, gamma (rel.)
    gam = envelope[-1][9]/datadat['mass'] + 1.0
    bet = math.sqrt( 1 - 1/gam**2 )
```

The dictionary `mod_pv` contains the phase and amplitude information that will be supplied to the simulation. The script executes `TRANSOPTR` using the `run_optr` command, saving the output data to the list `envelope`. This is then stored in string form and is written to file once the iterations are complete. The output files are named following the structure:

```
[grid]x[grid][tankName]_[optr-coordinate number].dat
```

2.1 Example Use: HEBT-11 MHz Buncher ($\mathbf{z}, \mathbf{P_z}$) Output

As an example, the plots in Figure 2 have been obtained by running `TRANSOPTR` a total of 5,625 times on the path shown in Figure 1. The output is a 75 by 75 grid spanning the voltage and phase configuration space domain of the buncher. What is shown on the left side of Figure 2 is actually $1/z$, the inverse `TRANSOPTR` bunch length. Further analysis of the implications of what is shown will be presented in a later report.

3 `pathFinder.py`

It is desired to record the (V_s, ϕ) coordinates of the traces shown in Figure 2 for accelerator tuning development at ISAC, though again this can be used for any two parameter sweep. The script

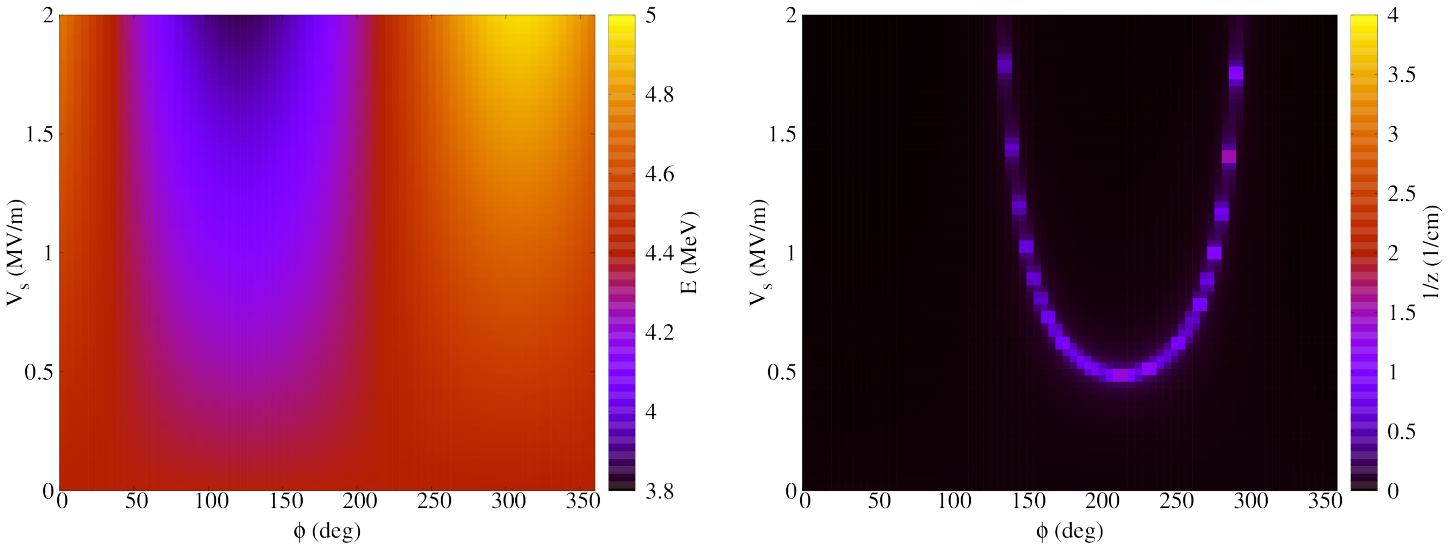

Figure 2: **Left:** `TRANSOPTR` computed HEBT11 MHz output energy, as measured at the DRAGON gas chamber for 11MHz HEBT buncher phase and longitudinal electric field, which represents the scaling factor applied to the z-electric field profile, shown in Ref. [1]. **Right:** 1/z (inverse bunch length), which is maximum when z = $\beta c \Delta t$ is minimized at the DRAGON gas chamber. **Voltage scaling factor does not directly denote the EPICS setpoint for the buncher amplitude.** Both scans are made on a 75 by 75 grid.

`pathFinder` [5] takes as input the files generated by `topology.py`. `pathFinder` analyzes the grid data which represents the `TRANSOPTR` output at the end of `fort.envelope`.

```
grid=len(rawData)
vOpt=[]
phiOpt=[]

#the lists vOpt and phiOpt contain the optimum voltage,phase pairs
for i in range(0,grid):
    datFrame = pd.DataFrame(rawData[i])[0]
    peaks = find_peaks(datFrame)[0]

    if(peaks.any()):
        #now I wanna store the v,phi coordinates of this peak in physical units
        #so I can use them for TRANSOPTR.
        peak = int(peaks[0]) # only accept 1st entry (there should be 1 anyway..)
vS = (peak/(grid-1))*(maxVolt-minVolt)
phi = (i/(grid-1))*(maxPhi-minPhi) + minPhi
        vOpt.append(vS)
        phiOpt.append(phi)
```

```
#make them into dataframes
voltDat = pd.DataFrame(vOpt)
phiDat = pd.DataFrame(phiOpt)
```

A `pandas` [6] dataframe (an array) is built for each line from the `topology` output file. The function `find_peaks` is imported from `scipy.signal` [7] and returns indices which correspond to the peak's index in the array. In the above example, the dataframe containing 1/z `TRANSOPTR` output at the DRAGON gas chamber is `datFrame` and the peak for each line is element `datFrame[peak]`. `find_peaks` detects peaks by simple comparison with neighboring points, as documented in its source code [8]:

```
Notes
-----
In the context of this function, a peak or local maximum is defined as any
sample whose two direct neighbours have a smaller amplitude. For flat peaks
(more than one sample of equal amplitude wide) the index of the middle
sample is returned (rounded down in case the number of samples is even).
For noisy signals the peak locations can be off because the noise might
change the position of local maxima. In those cases consider smoothing the
signal before searching for peaks or use other peak finding and fitting
methods (like ‘find_peaks_cwt‘).
```

3.1 Example Use: HEBT-11 MHz Buncher 1/z Peak Detection at DRAGON Gas Chamber

Examples of peak detection for the 11MHz HEBT buncher are shown in Figure 3, in which the same beampath from Figure 1 was used. At the top of the figure, a 75 by 75 scan was performed on the buncher. The dataset in the figure corresponds to phase step 30 of 75 for a voltage scan from 0 to 2 MV. This means that the phase is constant at 145.5° for each datapoint while the voltage is incremented across its full range. At the bottom of Figure 3, the resolution of the scan was lowered to 20 by 20. This is intended to demonstrate the importance of using a sufficiently large resolution if more precise results are desired. In both cases, `find_peaks` returns the maximum.

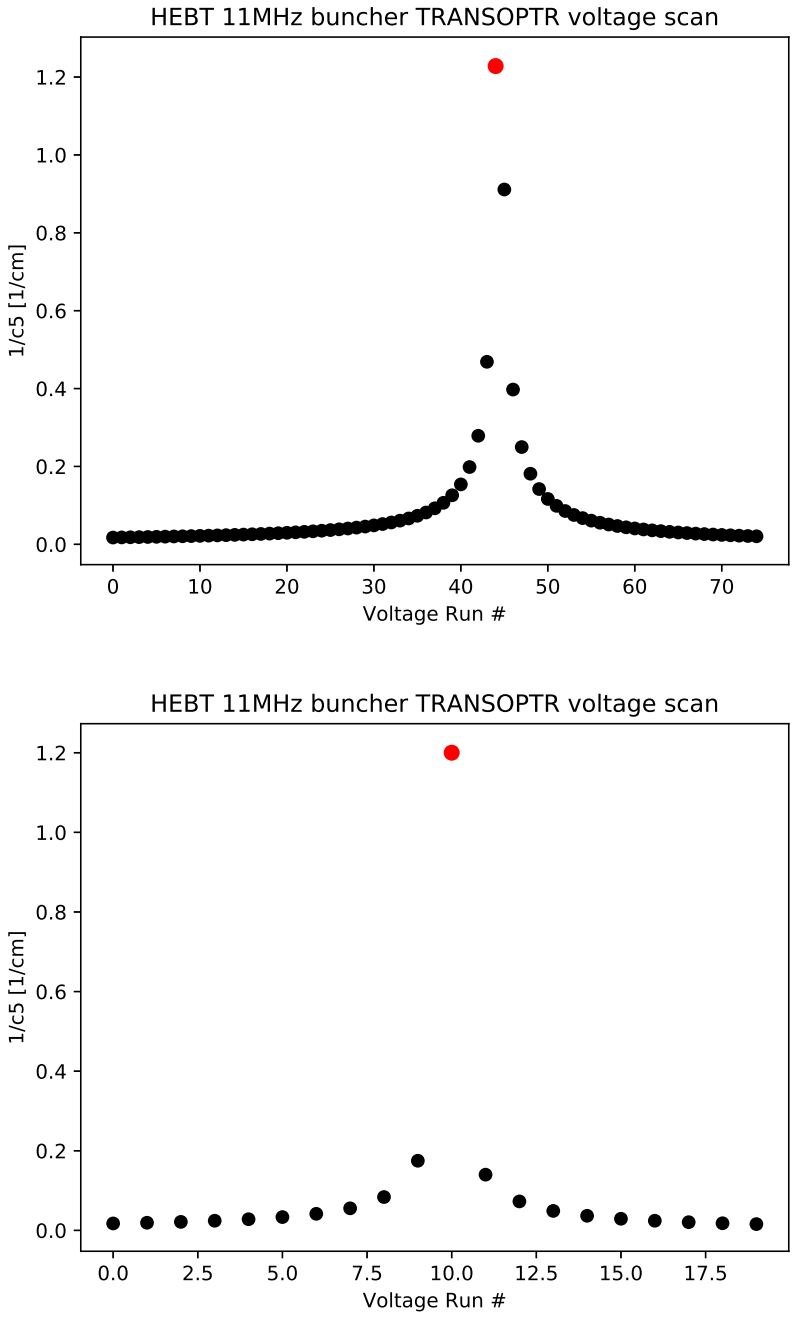

Figure 3: **Top:** Peak detection in `python` on `TRANSOPTR` 1/z curve for fixed phase, variable HEBT11 MHz buncher voltage. The x-axis units are model iterations, from 0 to 74, representing a 75x75 grid for the simulation used for this particular example. This run is for phase iteration 30 of 75 which corresponds to 145.95°. **Bottom:** Peak detection in `python` on `TRANSOPTR` 1/z curve for fixed phase, variable HEBT11 MHz buncher voltage. The x-axis units are model iterations, from 0 to 19, representing a 20x20 grid for the simulation used for this particular example. This run is for phase iteration 8 of 20 which corresponds to 144.00°. In both cases, the voltage is scanned from 0 to 2.0 MV/m

Using these pairs one can plot the optimum voltage and phase point combinations, which produce a minimized longitudinal bunch length at the location of the DRAGON gas chamber. This is shown in Figure 4. Its values represent the extracted peaks from the surface shown in Figure 2 on the right. This is the final output of `pathFinder.py`.

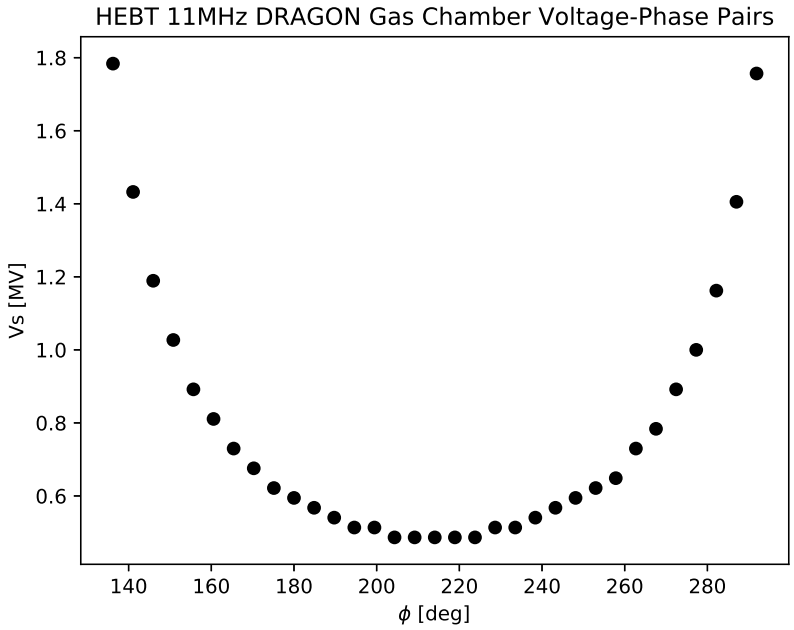

Figure 4: Optimum (V_s, ϕ) pairs for 11MHz HEBT buncher longitudinal (time) focus at DRAGON gas chamber, for 200 keV/u $^{22}Ne^{4+}$ beam.

4 Conclusion

This report details two `python` scripts which have been developed as part of the ongoing effort to extend `TRANSOPTR` for the study and operation of the ISAC linear accelerators. The script `topology.py` provides a useful tool for the running of two dimensional grid scans of arbitrary `TRANSOPTR` sequences. The companion script `pathFinder.py` performs a peak finding analysis on the output data produced by `topology`. This allows for the extraction of paths in the device configuration space on the surfaces generated by `TRANSOPTR`. An example case has been presented consisting of analyzing the 11MHz HEBT buncher's optimum voltage and phase pairs for a 200 keV/u $^{22}Ne^{4+}$ beam. This particular example could be of interest for DRAGON experiments.

References

[1] Olivier Shelbaya. TRANSOPTR Implementation of the HEBT Beamlines. Technical Report TRI-BN-19-06, TRIUMF, 2019.

[2] topology.py source from 2020-02-21 on isacdev04.triumf.ca.

[3] R. Laxdal. *Design Specification for ISAC HEBT*. Technical Report TRI-DN-99-23, TRIUMF, 1999.

[4] AK Mitra, PJ Bricault, IV Bylinskii, K Fong, G Dutto, RE Laxdal, RL Poirier, et al. *RF Test and Commissioning of the Radio Frequency Structures of the TRIUMF ISAC I facility*. In *Proceedings of LINAC*, page 106, 2002.

[5] pathFinder.py source from 2020-02-21 on isacdev04.triumf.ca.

[6] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[7] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[8] find_peaks sciPy GitHub source.

Appendices

A Path `ios-mws-hebt2-dragon.xml`

```
<?xml version='1.0' encoding='UTF-8' ?>
<beampath xmlns:xi='http://www.w3.org/2001/XInclude'>

<xi:include href='../sequence/ios_mws.xml' />
<xi:include href='../sequence/ios_db1.xml' />
<xi:include href='../sequence/ios_db10.xml' />
<xi:include href='../sequence/ilt_db33.xml' />
<xi:include href='../sequence/mebt_db0.xml' />
<xi:include href='../sequence/dtl_db0.xml' />
<xi:include href='../sequence/hebt_db0.xml' />
<xi:include href='../sequence/hebt_db9.xml' />
<xi:include href='../sequence/hebt2_db0.xml' />

</beampath>
```

B Tune design_hebt-dragon.xml

```
<root xmlns:xi='http://www.w3.org/2001/XInclude' path="ios-mws-hebt2-dragon">

<notes>These starting parameters correspond to what's in TRI-BN-19-06,
       which originally was coded in as a twissmatch.
       I've run it with vective(1) and extracted the sigma matrix
       elements to define the numbers below.

see conf. prof. below for output DTL long. emit
http://accelconf.web.cern.ch/AccelConf/pac97/papers/pdf/5W036.PDF
LANA simulations predict ez = 1.7 pi keV/u*ns (awful units)

November 24, 2019
Starting s-parameters are from TRI-BN-19-06 for HEBT-Q1 sequence.

</notes>
<optr s11="0.21435*cm" s22="6.44178*mrad" s33="0.543046*cm"
s44="9.23282*mrad" s55="1.345*mm" s66="25.171*mrad" r12="0.794"
r34="0.986" r56="-0.62" start="startOf_t3d_tune_hebt" end="DRAGON-Gas"/>

    <tune mass="21.9914*u" chargestate="4.0" energy="4.3983*MeV"/>

    <xi:include href='tune/ios-mws-hebt2-dragon_design200.xml'
    xpointer="xpointer(//root/tune)" parse="xml"/>

</root>
```

C Tune ios-mws-hebt2-dragon_design200.xml

```
<root acc="isac" branch="default" path="ios-mws-hebt2-dragon" tune="design">
  <tune chargestate="4.0" energy="4.39828*MeV" id="HEBT:XCB0" mass="21.9914*u">
    <set pv="HEBT:XCB0:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB0:CUR" value="0.0*A"/>
    <set pv="HEBT:Q1:CUR" value="9.72875*A"/>
    <set pv="HEBT:Q2:CUR" value="11.5374*A"/>
    <set pv="HEBT:XCB2:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB2:CUR" value="0.0*A"/>
    <set pv="HEBT:Q3:CUR" value="7.67904*A"/>
    <set pv="HEBT:Q5:CUR" value="7.02528*A"/>
```

```
  </tune>
  <tune chargestate="4.0" energy="4.39828*MeV" id="HEBT:STRP5" mass="21.9914*u">
    <set pv="HEBT:XCB5:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB5:CUR" value="0.0*A"/>
    <set pv="HEBT:Q6:CUR" value="10.72*A"/>
    <set pv="HEBT:Q7:CUR" value="4.64676*A"/>
    <set pv="HEBT:Q8:CUR" value="0*A"/>
    <set pv="HEBT:XCB8:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB8:CUR" value="0.0*A"/>
    <set pv="HEBT:Q9:CUR" value="4.8072*A"/>
    <set pv="HEBT:Q10:CUR" value="10.8061*A"/>
    <set pv="ISAC1:HEBT11:PHASE:SETPT" value="0*deg"/>
    <set pv="ISAC1:HEBT11:AMP:SETPT" value="549785.0*V"/>
    <set pv="ISAC1:HEBT35:PHASE:SETPT" value="0*deg"/>
    <set pv="ISAC1:HEBT35:AMP:SETPT" value="0.0*V"/>
    <set pv="HEBT:XCB10:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB10:CUR" value="0.0*A"/>
    <set pv="HEBT:Q11:CUR" value="12.8202*A"/>
    <set pv="HEBT:Q12:CUR" value="9.49825*A"/>
    <set pv="HEBT:XCB12:CUR" value="0.0*A"/>
    <set pv="HEBT:YCB12:CUR" value="0.0*A"/>
    <set pv="HEBT2:MB0:FLD:SETP" value="99867.574*G"/>
    <set pv="HEBT2:Q1:CUR" value="27.0591*A"/>
    <set pv="HEBT2:Q1:CUR" value="27.0591*A"/>
    <set pv="HEBT2:MB1:FLD:SETP" value="2903.088*G"/>
    <set pv="HEBT2:XCB2:CUR" value="0.0*A"/>
    <set pv="HEBT2:YCB2:CUR" value="0.0*A"/>
    <set pv="HEBT2:Q3:CUR" value="9.49825*A"/>
    <set pv="HEBT2:Q4:CUR" value="12.8202*A"/>
    <set pv="HEBT2:XCB4:CUR" value="0.0*A"/>
    <set pv="HEBT2:YCB4:CUR" value="0.0*A"/>
    <set pv="HEBT2:Q5:CUR" value="15.2051*A"/>
    <set pv="HEBT2:Q6:CUR" value="14.4659*A"/>
    <set pv="HEBT2:XCB6:CUR" value="-3.67919*A"/>
    <set pv="HEBT2:YCB6:CUR" value="6.689436*A"/>
    <set pv="HEBT2:Q7:CUR" value="7.62367*A"/>
    <set pv="HEBT2:Q8:CUR" value="0.725791*A"/>
  </tune>
</root>
```