New features in SixTrack: DYNK and DUMP

Kyrre Sjobak

High Luminosity LHC Week, October 30$^{\text{th}}$ 2015

Thanks to *Alessio Mereghetti* for original versions of DYNK and DUMP, *Riccardo De Maria* and *Andrea Santamaria* for some of the code development, and *Roderik Bruce*, *Hector Garcia Morales* and *Helmut Burkhardt* for many useful discussions.

Outline

What is DYNK

- Extension of SixTrack [1]
- Change element properties as a function of turn number
- Fully controlled by new block in `fort.3`;
  no need to change the sourcecode
- Supported elements/attributes:
  - All thin magnets (type ±1 — ±10)
    - Average multipole strength
  - RF cavities (type ±12)
    - Voltage
    - Harmonic number
    - Phase
  - Crab cavities (type ±23)
    - Voltage
    - Frequency
    - Phase
- Settings created using built-in "mini-programming-language" [2]

What is DYNK

- Extension of SixTrack [1]
- Change element properties as a function of turn number
- Fully controlled by new block in `fort.3`;
  no need to change the sourcecode
- Supported elements/attributes:
  - All thin magnets (type ± 1 — ± 10)
    - Average multipole strength
  - RF cavities (type ± 12)
    - Voltage
    - Harmonic number
    - Phase
  - Crab cavities (type ± 23)
    - Voltage
    - Frequency
    - Phase
- Settings created using built-in "mini-programming-language" [2]

What is DYNK

- Extension of SixTrack [1]
- Change element properties as a function of turn number
- Fully controlled by new block in `fort.3`;
  no need to change the sourcecode
- Supported elements/attributes:
  - All thin magnets (type ± 1 — ± 10)
    - Average multipole strength
  - RF cavities (type ± 12)
    - Voltage
    - Harmonic number
    - Phase
  - Crab cavities (type ± 23)
    - Voltage
    - Frequency
    - Phase
- Settings created using built-in
  "mini-programming-language" [2]

DYNK block in fort.3

Example:

```
DYNK
/ t_pi = t*pi
FUN pi CONST 3.14
FUN t TURN
FUN t_pi MUL pi t
/ Load myfile.txt
FUN myfile FILE myfile.txt
/ Apply myfile to some magnet,
/ starting at turn 5
SET dmqx2af5015+2 average_ms
   myfile 5 -1 0
/ Apply t_pi to a crab voltage
SET crab4 voltage t_pi 1 -1 0
NEXT
```

Verbose but straight-forward syntax.

Statement types:

FUN

`FUN name type arg1 arg2...`

Define a function which can be evaluated to provide a value for the element attributes.

SET

`SET element attribute function first last shift`

In the given time window, apply a function to the given element attributes.

DYNK example: RF Cavity detuning

- RF frequency shift ⇒ shift in revolution frequency ⇒ shift in beam momentum and orbit
- Used for measuring off-momentum lossmaps [3]

$V = V_0 \cos(\omega t); \quad \Phi(t) \equiv \omega t$

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \omega \equiv \omega_0 + \Delta\omega(t)$$

$$\Rightarrow \Phi(t) = \omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t$$

$$\Rightarrow V = V_0 \cos\left(\omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t\right)$$

Phase shift accumulates

- Linear frequency sweep:
  $\Delta\omega(t) = a \cdot t$, ($a$ is some constant)
  $\Delta\Phi(t) = \int_0^t \Delta\omega(t)\,\mathrm{d}t = \frac{a\omega\cdot t^2}{2}$.
- Can be implemented directly:
  – CODE –
- Or using numerical integration:
  – CODE –
- For large changes, remember wavelength: $t = T\Delta t + z/c$

DYNK example: RF Cavity detuning

- RF frequency shift $\Rightarrow$ shift in revolution frequency $\Rightarrow$ shift in beam momentum and orbit
- Used for measuring off-momentum lossmaps [3]

$$V = V_0 \cos(\omega t); \quad \Phi(t) \equiv \omega t$$

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \omega \equiv \omega_0 + \Delta\omega(t)$$

$$\Rightarrow \Phi(t) = \omega_0 t + \int_0^t \Delta\omega(t')\ \mathrm{d}t$$

$$\Rightarrow V = V_0 \cos\left(\omega_0 t + \int_0^t \Delta\omega(t')\ \mathrm{d}t\right)$$

Phase shift accumulates

- Linear frequency sweep: $\Delta\omega(t) = a \cdot t$, ($a$ is some constant) $\Delta\Phi(t) = \int_0^t \Delta\omega(t)\ \mathrm{d}t = \frac{a\omega \cdot t^2}{2}$.
- Can be implemented directly:
  – CODE –
- Or using numerical integration:
  – CODE –
- For large changes, remember wavelength: $t = T\Delta t + z/c$

DYNK example: RF Cavity detuning

- RF frequency shift $\Rightarrow$ shift in revolution frequency $\Rightarrow$ shift in beam momentum and orbit
- Used for measuring off-momentum lossmaps [3]

$$V = V_0 \cos(\omega t); \quad \Phi(t) \equiv \omega t$$

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \omega \equiv \omega_0 + \Delta\omega(t)$$

$$\Rightarrow \Phi(t) = \omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t$$

$$\Rightarrow V = V_0 \cos\left(\omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t\right)$$

Phase shift accumulates

- Linear frequency sweep:
  $\Delta\omega(t) = a \cdot t$, ($a$ is some constant)
  $\Delta\Phi(t) = \int_0^t \Delta\omega(t)\,\mathrm{d}t = \frac{a\omega \cdot t^2}{2}$.
- Can be implemented directly:

fort.3

```
FUN phase quad a/2 0 0
SET acsca.d5l1.b1 lag_angle phase 1 -1 0
```

- Or using numerical integration: - CODE -
- For large changes, remember wavelength: $t = T\Delta t + z/c$

DYNK example: RF Cavity detuning

- RF frequency shift $\Rightarrow$ shift in revolution frequency $\Rightarrow$ shift in beam momentum and orbit
- Used for measuring off-momentum lossmaps [3]

$$V = V_0 \cos(\omega t); \quad \Phi(t) \equiv \omega t$$

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \omega \equiv \omega_0 + \Delta\omega(t)$$

$$\Rightarrow \Phi(t) = \omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t$$

$$\Rightarrow V = V_0 \cos\left(\omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t\right)$$

Phase shift accumulates

- Linear frequency sweep:
  $\Delta\omega(t) = a \cdot t$, (*a* is some constant)
  $\Delta\Phi(t) = \int_0^t \Delta\omega(t)\,\mathrm{d}t = \frac{a\omega \cdot t^2}{2}$.
- Can be implemented directly:
  - CODE -
- Or using numerical integration:

fort.3

```
FUN deltaFreq LIN a 0
//Convert the frequencies from MHz to radians/turn/Hz
FUN HzInvTurnFactor CONST 5.587288765e-4
FUN deltaPhi MUL deltaFreq HzInvTurnFactor
// Phi_turn = deltaW + Phi_turn-1
FUN phi_c2a IIR 1 IIRcoeffs.txt deltaPhi
//Set the phases
SET CRAB2A phase phi_c2a 1 -1 0
```

IIRcoeffs.txt

```
0 1 0 0 0
1 0 0 1 0
```

DYNK example: RF Cavity detuning

- RF frequency shift $\Rightarrow$ shift in revolution frequency $\Rightarrow$ shift in beam momentum and orbit
- Used for measuring off-momentum lossmaps [3]

$$V = V_0 \cos(\omega t); \quad \Phi(t) \equiv \omega t$$

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \omega \equiv \omega_0 + \Delta\omega(t)$$

$$\Rightarrow \Phi(t) = \omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t$$

$$\Rightarrow V = V_0 \cos\left(\omega_0 t + \int_0^t \Delta\omega(t')\,\mathrm{d}t\right)$$

Phase shift accumulates

- Linear frequency sweep: $\Delta\omega(t) = a \cdot t$, (a is some constant) $\Delta\Phi(t) = \int_0^t \Delta\omega(t)\,\mathrm{d}t = \frac{a\omega \cdot t^2}{2}$.
- Can be implemented directly:
  - CODE -
- Or using numerical integration:
  - CODE -
  - General: Replace `FUN deltaFreq ...` with any expression
  - Usable only for slowly changing frequencies, otherwise too inaccurate
- For large changes, remember wavelength: $t = T\Delta t + z/c$

Output file dynksets.dat

- The settings of every element/attribute pair affected by DYNK is for *all* turns written to a file
- Useful for debugging your "program"
- Or making plots for presentations etc. . .
- Can be turned off with flag `NOFILE`
- Format:
  `turn element attribute SETidx funname value`
- Example program for parsing/plotting:
  https://github.com/kyrsjo/SixtrackTools/blob/master/analysis/analyze_dynksets.py

Example dynksets.dat (reduced):

```
# turn element attribute SETidx funname value
1 CRAB2A phase   -1 N/A          0.000000000E+00
1 CRAB2A voltage  5 off          0.000000000E+00
2 CRAB2A voltage  9 on_2a        0.242183208E+01
6 CRAB2A phase   13 quenchPhase  0.000000000E+00
6 CRAB2A voltage 17 v_c2v        0.242183208E+01
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
```

Output file dynksets.dat

- The settings of every element/attribute pair affected by DYNK is for *all* turns written to a file
- Useful for debugging your "program"
- Or making plots for presentations etc. . .
- Can be turned off with flag `NOFILE`
- Format:
  `turn element attribute SETidx funname value`
- Example program for parsing/plotting:
  https://github.com/kyrsjo/SixtrackTools/blob/master/analysis/analyze_dynksets.py

Example dynksets.dat (reduced):

```
# turn element attribute SETidx funname value
1 CRAB2A phase   -1 N/A          0.000000000E+00
1 CRAB2A voltage  5 off          0.000000000E+00
2 CRAB2A voltage  9 on_2a        0.242183208E+01
6 CRAB2A phase   13 quenchPhase  0.000000000E+00
6 CRAB2A voltage 17 v_c2v        0.242183208E+01
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
```

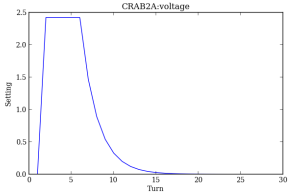
Output file dynksets.dat

- The settings of every element/attribute pair affected by DYNK is for *all* turns written to a file
- Useful for debugging your "program"
- Or making plots for presentations etc. . .
- Can be turned off with flag `NOFILE`
- Format:
  `turn element attribute SETidx funname value`
- Example program for parsing/plotting:

  https://github.com/kyrsjo/SixtrackTools/blob/master/analysis/analyze_dynksets.py

Example dynksets.dat (reduced):

```
# turn element attribute SETidx funname value
1 CRAB2A phase   -1 N/A           0.000000000E+00
1 CRAB2A voltage  5 off           0.000000000E+00
2 CRAB2A voltage  9 on_2a         0.242183208E+01
6 CRAB2A phase   13 quenchPhase   0.000000000E+00
6 CRAB2A voltage 17 v_c2v         0.242183208E+01
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
```



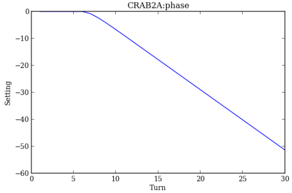
Output file dynksets.dat

- The settings of every element/attribute pair affected by DYNK is for *all* turns written to a file
- Useful for debugging your "program"
- Or making plots for presentations etc. . .
- Can be turned off with flag `NOFILE`
- Format:
  `turn element attribute SETidx funname value`
- Example program for parsing/plotting:

https://github.com/kyrsjo/SixtrackTools/blob/master/analysis/analyze_dynksets.py

Example dynksets.dat (reduced):

```
# turn element attribute SETidx funname value
1 CRAB2A phase   -1 N/A          0.000000000E+00
1 CRAB2A voltage  5 off          0.000000000E+00
2 CRAB2A voltage  9 on_2a        0.242183208E+01
6 CRAB2A phase   13 quenchPhase  0.000000000E+00
6 CRAB2A voltage 17 v_c2v        0.242183208E+01
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
7 CRAB2A phase   13 quenchPhase -0.822175200E+00
```



DYNK example: Magnet ripple

- Apply a sinusiodal ripple on top of the magnet strength:

$$y(t) = A\cos\left(\frac{2\pi(t-1)}{\text{period}} + \phi\right)$$

- Old input block RIPP can be converted directly (and automatically):

```
RIPPLE OF POWER SUPPLIES
dmqx1f5015+2 3.2315D-10 224.9
NEXT
```

```
DYNK
NOFILE
FUN RIPP-dmqx1f5015+2 COSF_RIPP 3.2315D-10 224.9 0.0
SET dmqx1f5015+2 average_ms RIPP-dmqx1f5015+2 1 -1 0
NEXT
```

- Perfectly reproduces results from the old block
- Typically used for long dynamic apperture studies
  $\Rightarrow$ use NOFILE flag

DUMPing particle data

- Extract the beam population at (a) given element(s)
- Controlled by a block in `fort.3`
- Syntax:
  `elementName frequency unitNum formatIdx (filename)`
  - Element name should be a single element not in a BLOC
- Example:
  ```
  DUMP
  ip1 1 660 2 IP1_DUMP.dat
  NEXT
  ```
- Extra options:
  - `FRONT`
  - `HIGH`
  - Element name = `ALL`

DUMPing particle data

- Extract the beam population at (a) given element(s)
- Controlled by a block in `fort.3`
- Syntax:
  `elementName frequency unitNum formatIdx (filename)`
  - Element name should be a single element not in a BLOC
- Example:

```
DUMP
ip1 1 660 2 IP1_DUMP.dat
NEXT
```

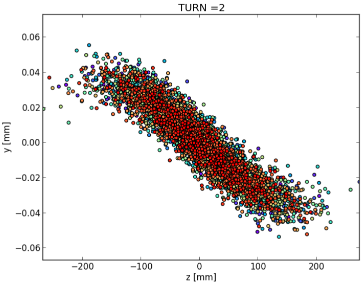
- Extra options:
  - `FRONT`
  - `HIGH`
  - Element name = `ALL`

DUMPing particle data

- Extract the beam population at (a) given element(s)
- Controlled by a block in `fort.3`
- Syntax:
  `elementName frequency unitNum formatIdx (filename)`
  - Element name should be a single element not in a BLOC
- Example:
```
DUMP
ip1 1 660 2 IP1_DUMP.dat
NEXT
```
- Extra options:
  - FRONT
  - HIGH
  - Element name = ALL

IP1_DUMP.dat:

```
# DUMP format #2, bez=ip1 , dump period= 1
# ID turn s[m] x[mm] xp[mrad] y[mm] yp[mrad]
z[mm] dE/E ktrack
1 1 0.00000 1.201547790E-02 -7.793418012E-02
-1.784574172E-02 3.023749796E-01
-7.344122000E+01 7.428571429E-06 31
```
etc.

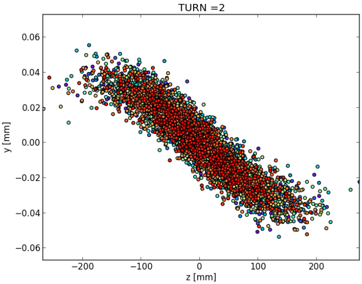


DUMPing particle data

- Extract the beam population at (a) given element(s)
- Controlled by a block in `fort.3`
- Syntax:
  `elementName frequency unitNum formatIdx (filename)`
  - Element name should be a single element not in a BLOC
- Example:
  ```
  DUMP
  ip1 1 660 2 IP1_DUMP.dat
  NEXT
  ```
- Extra options:
  - `FRONT`
  - `HIGH`
  - Element name = `ALL`

IP1_DUMP.dat:

```
# DUMP format #2, bez=ip1 , dump period= 1
# ID turn s[m] x[mm] xp[mrad] y[mm] yp[mrad]
z[mm] dE/E ktrack
1 1 0.00000 1.201547790E-02 -7.793418012E-02
-1.784574172E-02 3.023749796E-01
-7.344122000E+01 7.428571429E-06 31
```
etc.



Future: Unified output facilities

- **Problem:** Current output mechanisms quite diverse / obscure
- **Proposal:** Unify these into a common `OUTPUT` block, based on ideas from the current `DUMP` and `DYNK`
  - Should handle both pre-run output and tracking output
  - Hooks for analysis blocks etc. to enable outputs
  - Remove one of the particle number restrictions...
- **Obstacle:** No clear scheme for fortran file unit # allocation
  - Identify range of safe IDs, make allocation/manager function.
  - MadX has a similar mechanism.

Future: Unified output facilities

- **Problem:** Current output mechanisms quite diverse / obscure
- **Proposal:** Unify these into a common `OUTPUT` block, based on ideas from the current `DUMP` and `DYNK`
  - Should handle both pre-run output and tracking output
  - Hooks for analysis blocks etc. to enable outputs
  - Remove one of the particle number restrictions...
- **Obstacle:** No clear scheme for fortran file unit # allocation
  - Identify range of safe IDs, make allocation/manager function.
  - MadX has a similar mechanism.

Future: Unified output facilities

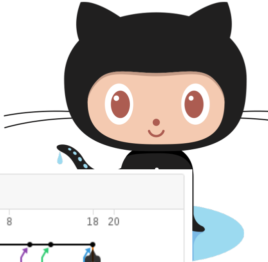
- **Problem:** Current output mechanisms quite diverse / obscure
- **Proposal:** Unify these into a common `OUTPUT` block, based on ideas from the current `DUMP` and `DYNK`
  - Should handle both pre-run output and tracking output
  - Hooks for analysis blocks etc. to enable outputs
  - Remove one of the particle number restrictions. . .
- **Obstacle:** No clear scheme for fortran file unit # allocation
  - Identify range of safe IDs, make allocation/manager function.
  - MadX has a similar mechanism.

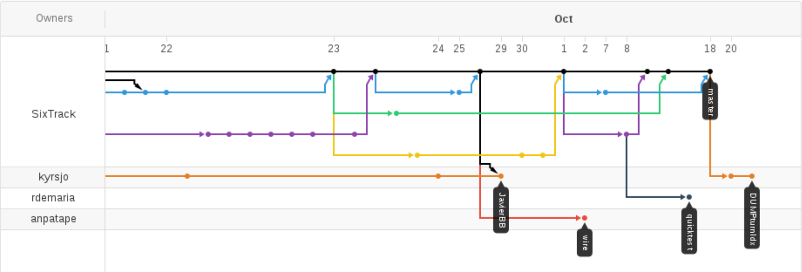
Distribution of SixTrack via GIT

- GIT has replaced SVN for version control
  - Much better support for parallel development, merging branches is easy
  - The most widely used version control tool today
  - Distributed: All users have full copy of the history
- Repository hosted on GitHub
  - `https://github.com/SixTrack/SixTrack`
  - Recomended by CERN for open projects [4]
  - Allows non-CERN collaborators
  - Makes development easier and more structured: Issue tracker, pull requests...

Distribution of SixTrack via GIT

- GIT has replaced SVN for version control
  - Much better support for parallel development, merging branches is easy
  - The most widely used version control tool today
  - Distributed: All users have full copy of the history
- Repository hosted on GitHub





Currently in very active development

Summary and conclusions

DYNK

- Mechanism for changing element properties as a function of time
- Very useful for studying fast failure scenarios
- Replaces older `RIPP` mechanism and multiple private hacks
- Easy to extend to support new function and/or element types

DUMP

- Extracting the beam population for analysis outside SixTrack
- Might be the seed for remodelling the output system

Source distribution / GIT

- SixTrack is now distributed via GitHub
- Makes coordination much easier
- Avoid "monster-commits"

Thanks!

(More) Questions?

Bibliography I

[1] K. Sjobak , H. Burkhardt, R. De Maria, A. Mereghetti, A. Santamaría García: *GENERAL FUNCTIONALITY FOR TURN-DEPENDENT ELEMENT PROPERTIES IN SIXTRACK*; IPAC'15.

[2] Kyrre Sjobak and Frank Schmidth: *SixTrack User's reference manual, v2.5.28*. https://github.com/SixTrack/SixTrack/raw/master/Doc/user_manual/six.pdf

[3] Hetor Garcia Morales et al: *LHC off-momentum collimation simulation (work in progress)*; LHC Collimation Working Group meeting #194, 21/9/2015. https://indico.cern.ch/event/446488/other-view?view=standard

Bibliography II

[4] *KB0003132: When is it appropriate to use CERN GitLab or external services such as Github?*
https://cern.service-now.com/service-portal/article.do?n=KB0003132&s=gitlab

Implementation – data structure

Data stored in common blocks defined in block `comdynk`

- FUN statements stored in one table
  - 1 row per FUN
  - Columns: Name (index in cexpr_dynk), function type, 3×free
  - Arrays available with “allocatable” memory, storing integers, reals and strings

```
integer funcs_dynk (maxfuncs_dynk,5)              !Main FUN table
integer iexpr_dynk (maxdata_dynk)                 !Free storage (integers)
double precision fexpr_dynk (maxdata_dynk)        !Free storage (doubles)
character(maxstrlen_dynk) cexpr_dynk(maxdata_dynk)!Free storage (strings)

integer nfuncs_dynk, niexpr_dynk, nfexpr_dynk, ncexpr_dynk !Allocation
```

Implementation – data structure

Data stored in common blocks defined in block comdynk

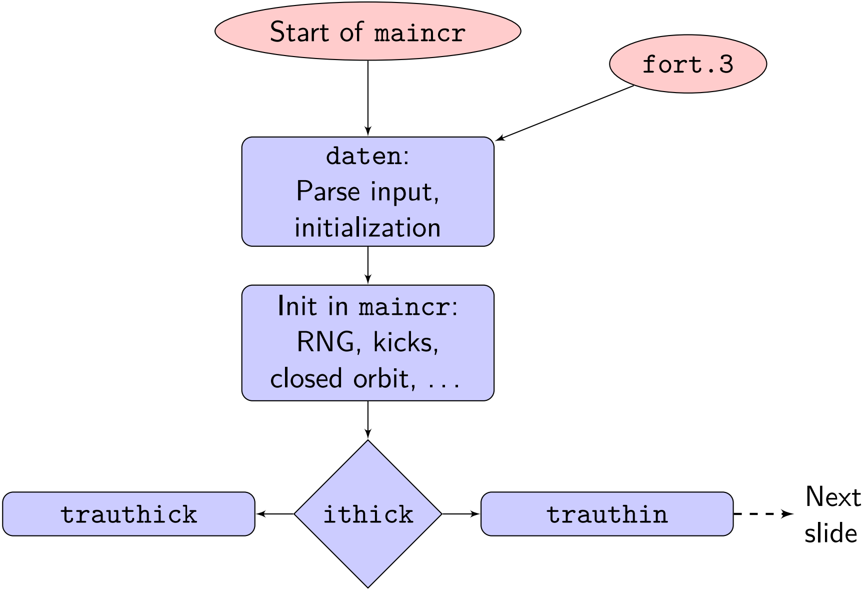
- FUN statements stored in one table
  - 1 row per FUN
  - Columns: Name (index in cexpr_dynk), function type, 3×free
  - Arrays available with "allocatable" memory, storing integers, reals and strings

```
integer funcs_dynk (maxfuncs_dynk,5)                 !Main FUN table
integer iexpr_dynk (maxdata_dynk)                    !Free storage (integers)
double precision fexpr_dynk (maxdata_dynk)           !Free storage (doubles)
character(maxstrlen_dynk) cexpr_dynk(maxdata_dynk)!Free storage (strings)

integer nfuncs_dynk, niexpr_dynk, nfexpr_dynk, ncexpr_dynk !Allocation
```

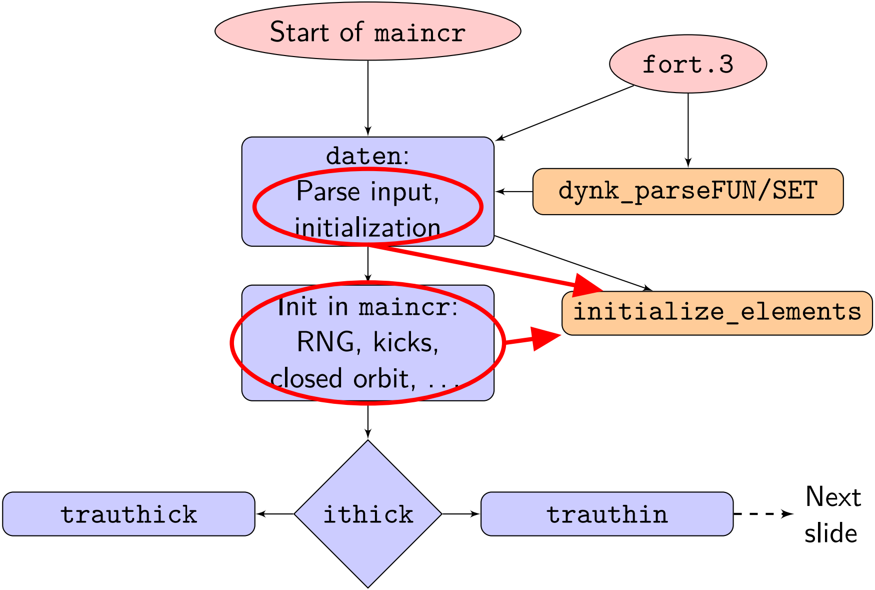
- Two similar tables for SET statements
  - Columns: Function index, turn limits, turn shift
  - Columns: Element name, attribute name
  - Also store pre-tracking values

```
integer sets_dynk(maxsets_dynk, 4)                    !  SET table (ints)
character(maxstrlen_dynk) csets_dynk (maxsets_dynk,2) !  SET table (names)
integer nsets_dynk

character(maxstrlen_dynk) csets_unique_dynk (maxsets_dynk,2) !  Store the pre-tracking
double precision fsets_origvalue_dynk(maxsets_dynk)          !  settings from fort.2
```

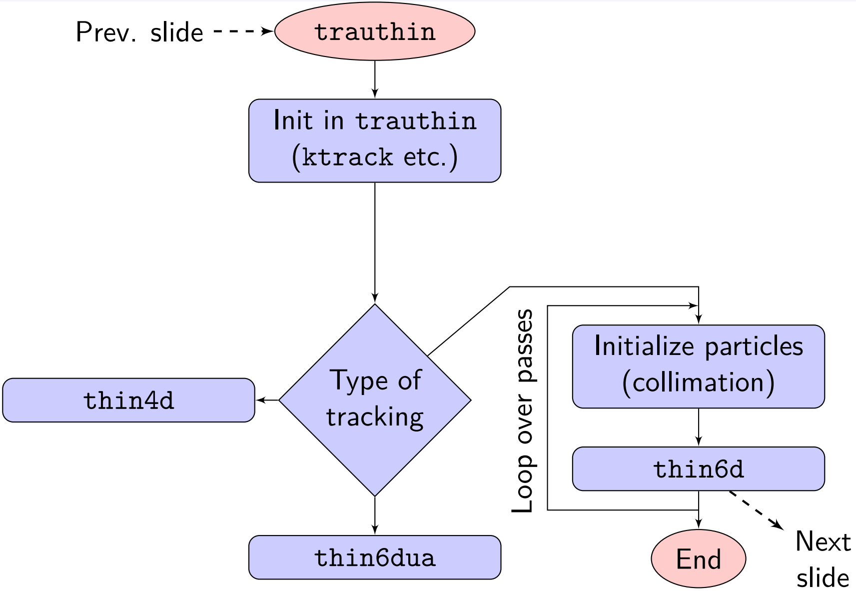
Implementation – program flow (1/3)



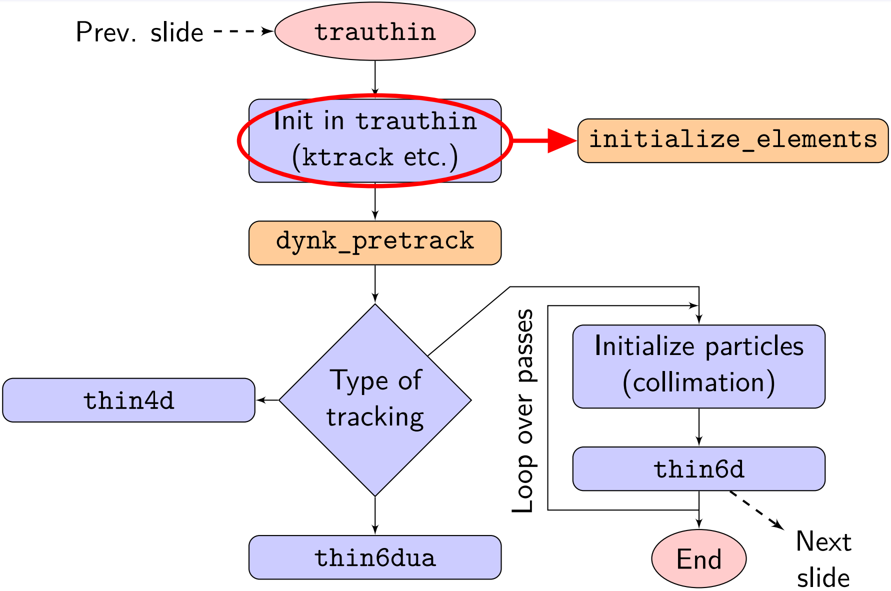
Implementation – program flow (1/3)



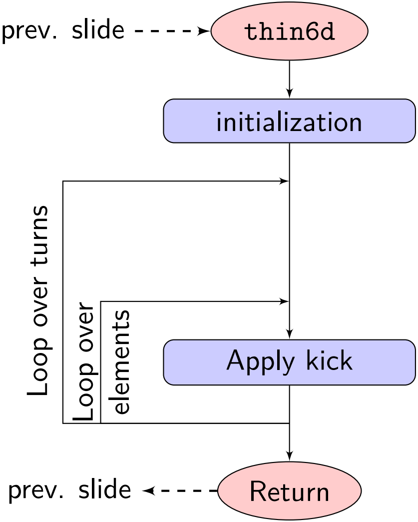
Implementation – program flow (2/3)



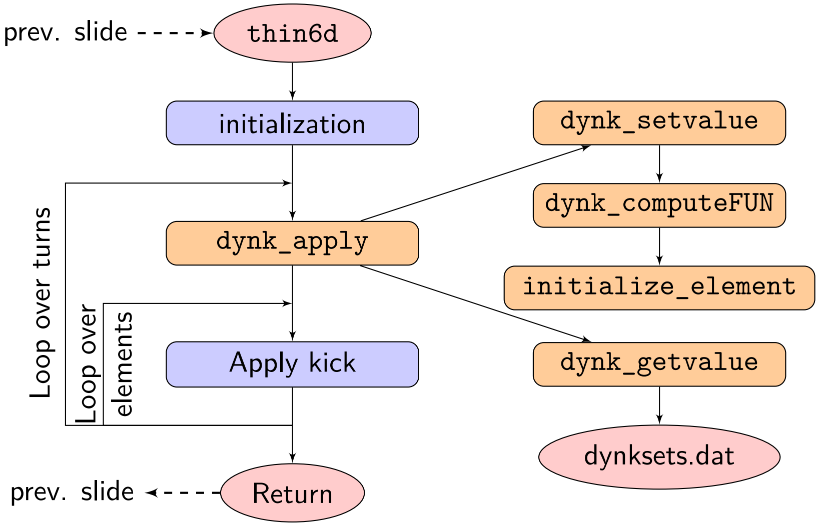
Implementation – program flow (2/3)



Implementation – program flow (3/3)



Implementation – program flow (3/3)



Implementation – Adding new FUNs

In principle

- Allocate a function index
- Add code to the functions `dynk_computeFUN` and `dynk_parseFUN`

Implementation – Adding new FUNs

In principle

- Allocate a function index
- Add code to the functions `dynk_computeFUN` and `dynk_parseFUN`

Code example – evaluation:

```
recursive double precision function
& dynk_computeFUN( funNum, turn ) result(retval)
...
select case ( funcs_dynk(funNum,2) )
...
case( 32 )
retval = sin( dynk_computeFUN(
&              funcs_dynk(funNum,3),turn) )
...
end select
end function
```

Code example – initialization: FUN SIN <function name>

```
subroutine dynk_parseFUN( getfields_fields,
& getfields_lfields,getfields_nfields )
...
select case ( getfields_fields(3)
& (1:getfields_lfields(3)) )
...
case("SIN")[a]
!  DATA: Name, Type, function index, -, -
funcs_dynk(nfuncs_dynk,1) = ncexpr_dynk
funcs_dynk(nfuncs_dynk,2) = 32
funcs_dynk(nfuncs_dynk,3) =
&    dynk_findFUNindex(getfields_fields(4)
&    (1:getfields_lfields(4)), 1)

!  NAME
cexpr_dynk(ncexpr_dynk)(1:getfields_lfields(2))
& = getfields_fields(2)(1:getfields_lfields(2))
...
end select
end subroutine
```

[a] Some boilerplate code, incl. input sanity checks, is omitted

Implementation – Adding new elements / attributes

- Add the element to `dynk_setvalue` and `dynk_getvalue`
- If the element uses data from other variables than `ed`, `ek` and `el` for kicking:
  Add code to `initialize_element`
- Sometimes ugly interactions occur...
  - Initialization spread thin throughout the code
  - Other elements depending (indirectly) on this setting

Our development model

Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.

Our development model

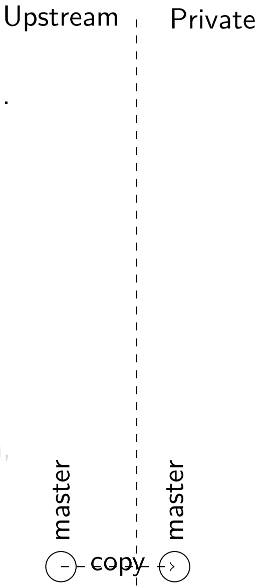
Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



Our development model

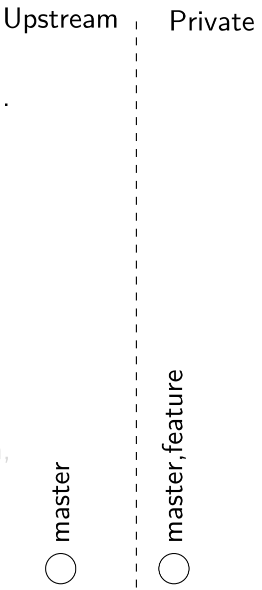
Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



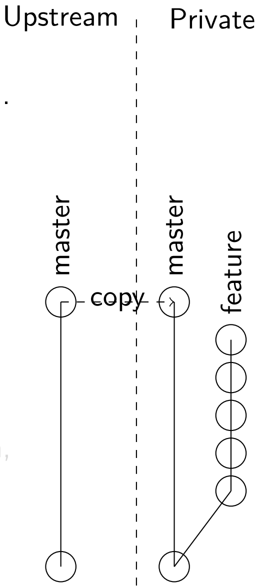
Our development model

Master repository:
`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



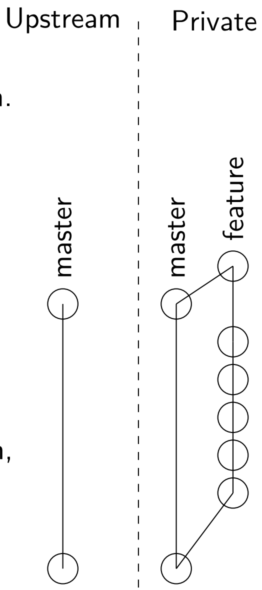
Our development model

Master repository:
`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



Our development model

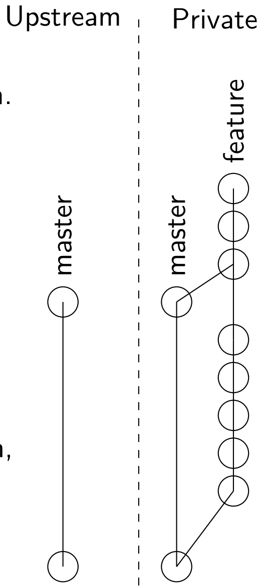
Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



Our development model

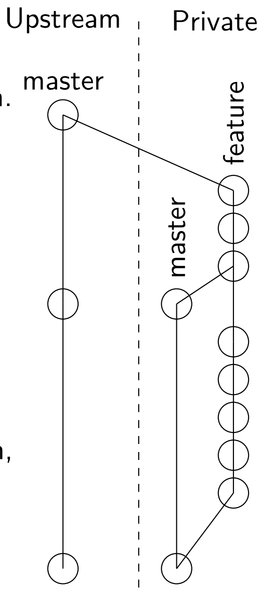
Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



Our development model

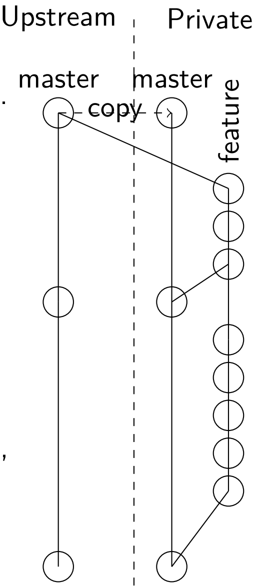
Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.



Our development model

Master repository:

`https://github.com/SixTrack/SixTrack`

- Contains the released versions in its `master` branch.
- Repository maintained by Ricardo and Kyrre.

Personal repositories:

- To add a new feature, create a *fork* on GitHub.
- In the newly created personal repository, create a branch.
- Do your changes in this branch, while your `master` tracks the upstream `master`.
- To get the changes from upstream into your branch, merge upstream's master into your branch
- To merge your code into upstream (for a new release), create a *pull request* on GitHub.