

EFDA–JET–CP(08)06-04

E. Giovannozzi
and JET EFDA contributors

Object-Oriented Interactive Access to JET Database

“This document is intended for publication in the open literature. It is made available on the understanding that it may not be further circulated and extracts or references may not be published prior to publication of the original when applicable, or without the consent of the Publications Officer, EFDA, Culham Science Centre, Abingdon, Oxon, OX14 3DB, UK.”

“Enquiries about Copyright and reproduction should be addressed to the Publications Officer, EFDA, Culham Science Centre, Abingdon, Oxon, OX14 3DB, UK.”

Object-Oriented Interactive Access to JET Database

E. Giovannozzi and JET EFDA contributors*

JET-EFDA, Culham Science Centre, OX14 3DB, Abingdon, UK

¹*Associazione EURATOM-ENEA sulla fusione, CR Frascati, Roma, Italy,*

** See annex of M.L. Watkins et al, "Overview of JET Results",
(Proc. 21st IAEA Fusion Energy Conference, Chengdu, China (2006)).*

Preprint of Paper to be submitted for publication in Proceedings of the
8th International FLINS Conference on Computer Intelligence in Decision and Control, Madrid, Spain.
(21st September 2008 - 24th September 2008)

ABSTRACT.

Some preliminary classes have been developed in MATLAB to interface with the JET PPF system which stores the data elaborated after the discharge. These classes simplify the most common operations required during an interactive analysis and visualization of JET data, while keeping all the power of a full programming language. The first class is `ppfs` which given a shot number retrieves a list of all the main family (DDAs) related to that shot, the second is `ddas` which lists all the DTYPEs belonging to a DDA. The last two classes are used for retrieving the actual data and perform the most common operation, such as basic plotting slicing and arithmetic operation.

1. INTRODUCTION

Object Oriented (OO) languages are widely used as they permit to write large codes in a more natural and easy way. Many languages have been changed in order to support a form or another of objects [1].

An interactive environment, and language, as MATLAB[†] has the possibility to manipulate natively arrays and, with OO features, other kind of data while retaining all the strength of a full programming language. In the following MATLAB has been used to develop four classes to retrieve data from the JET PPF (Processed Pulse File) system, a porting of these classes to other interactive environment is also under way. The jet PPF system [2] stores data elaborated after every discharge. Every signal stored there is characterized by its family name DDA (Diagnostic Data Areas) and its proper name DTYPE (Data Type), each of them of four characters. They are also characterized by the owner (in the case of private PPFs) and a sequence number (we are not caring about the sequence number now, but we are assuming to be interested only in the last written sequence number). An owner (by default JETPPF), a DDA and a DTYPE is all that is required to get the proper data. All the data are basically bi-dimensional, matrices with a time and a radial coordinate, even though one of them could be missing if one is interested in one-dimensional data. A short description of the data is also stored in the archive. There are three basic operations that the interactive user might want to do, the first is to retrieve the correct name of a given measurable quantity, the second to make slices and plot them, third to make operations between different data. This has been achieved with four classes. The first two are directly related to the DDA and DTYPE parameters, the first class called `ppfs` give a list of the DDAs, while the second class called `ddas` gives a list of a DTYPE inside a DDA. The second two classes are more related to the data itself, a first class to deal with the bi-dimensional data, the second to deal with the slices. A single class could have been used instead of two different classes, and in the future they may be combined together.

2. PPFs AND DDAS CLASSES

These two classes are quite similar and in the following the `ppfs` class is described first. From the user point of view, a `ppfs` object appears as a structure where its fields are the DDAs, and consequently the DTYPEs appear as fields of DDAs. This has been obtained overloading the field operator (the function `subsref` of the object). Even though there is strict separation in MATLAB

among input and output parameter of a function, it is possible, with a perfectly legal trick (using only documented functions), to change an input parameter. This is specially important with OO programming for interactive use as it is more natural and fast to have objects that can change their status depending on the methods called. Even if the access to JET archive is quite fast the retrieved data are stored in memory changing the `ddas` or the `ppfs` objects accordingly.

The two main attributes of the `ppfs` class are two arrays, one is a character array with the names of all the `DDAs`, the second is a cell array where the `ddas` object would be stored. There are other attributes like the shot number or the `userid` for private PPFs. The `ddas` class is similar, with corresponding arrays for storing the `DTYPE` names and objects.

The main constructor fetches the `DDA` names from JET archive and returns to the caller. No `ddas` object is stored in the second array. This is the aim of the `subsref` method. When a complete field is referenced, the `subsref` method calls a constructor for the corresponding `DDA`, which in turn fetches the list of the `DTYPE`. The `subsref` function would then modify the original `ppfs` object, so that a following reference to the same field does not result in a new access to the database. The `subsref` method would then return a `ddas` object (that accordingly shows the list of its `DTYPE`). If one asks an incomplete field then a `ppfs` object, with only the `DDAs` beginning with that initials is returned (that would then display on the screen only the selected `DTYPE`). The `ddas` objects have a similar behaviour to the `ppfs` objects with `v2d` objects in place of `ddas` objects. The other two methods are: the `display` method which display on the screen the list of `DDA` (`DTYPE`) belonging to the `ppfs` (`ddas`) object; and a `list` method to convert a `ppfs` (`ddas`) object to a list of names.

3. V2D CLASS

The data in the PPF system are basically bidimensional, with the first dimension being a spatial dimension and the second one a temporal dimension. One of the two dimensions could be absent, in this case the data depend only on time (by far the most frequent case), or on spatial position. The `v2d` class has been written in order to resemble as much as possible this situation.

There are three fields: the first is r , a spatial coordinate, the second is t , a time coordinate and the third is v , the value of the data at the specific time and radius. The radial coordinates could be a matrix, while the time coordinates is always a vector. This way is possible to model a situation (for example the ECE temperature) where the detectors measure the temperature at a position which depends on the magnetic field.

The most common operation required is slicing the data along a dimension, and two methods have been written to slice the data along time or space. These methods accept a parameter for selecting the position of the slice. For example, to get a slice along time, a parameter with the radial position should be supplied. There are two kinds of slicing: in the first one the closest profile to the given position or time is returned, in the second one an interpolation is performed. This kind of slicing is mostly useful when an object `v2d` has got an r coordinate which is a matrix, the first kind of slicing would give jumps when a different detector is selected. As an option for the first kind of slicing there is the possibility to follow one detector even if its position is changing.

The first kind of slicing could also return all the possible slices if no parameter is given, or an array of slices if the supplied parameter is a vector. The slices are objects of `vxy` class. Array of `vxy` objects can be created and a multiple slice produces an array of `vxy` object.

Actually it would have been possible to use a single class instead of two classes but in this preliminary version of the code two classes were written as the `vxy` class was already present (it was used to access experimental data in FTU which are mostly one-dimensional); moreover there are simple methods in the `vxy` class that would have required a different design if they had to be applied to bidimensional data.

4. VXY CLASS

The data of a `vxy` object are one-dimensional, with two arrays `x`, `y`. A further field `ud` is used to store additional data. Even if it is not required the `x` being in ascending or descending order, many methods that imply an interpolation work only for monotonic `x` data. There are four main methods:

- Methods that perform an unary or a binary operation between two `vxy` objects mostly implemented as operators, or simple function.
- Methods that change the object, like swapping the `x` and the `y` field or restricting the `x` between two given values, or getting interpolated value at a given coordinate.
- Methods to get the `x` and the `y` array or the `ud` field.
- Method `plot` which plots arrays of `vxy` object.

Some of the methods are implemented as pseudo-fields overloading the `subsref` method, for example the method which get the `x` or `y` array. They appear to the user as fields of a structure, while the `subsref` is actually called. This is useful as it hides the true fields from the user. The methods that combine two different `vxy` objects need a common `x` base where to interpolate the data. The default choice is to get the union of the two `x` bases in the zone where both were defined. The interpolation is linear but other kind of interpolation and `x` base choice could be selected.

5. EXAMPLES

This is a simple example showing from the user point of view the interaction with the system. First the list of all the public DDA belonging to the Pulse No: 57941 is printed on the screen (only few of them are shown here).

```
>> w = ppfs(57941)
w =
  Shot: 57941
      uid: JETPPF
  n.DDA: 106
      bol4: - 7
      bolo: - 7
      ...
```

Then the list of all the DTYPE belonging to the `nft2` DDA and starting with `pr` is shown on the screen. A small comment attached to the DTYPE is also shown, so the users can quickly find what they are searching for.

```
>> w.nft2.pr
ans =
  Shot: 57941
  dda: nft2
  uid: JETPPF
n. DT: 3
  prhi: 21 x 145 - ne(sqrt(psi)) Hi Limit
  prlo: 21 x 145 - ne(sqrt(psi)) Low Limit
  prof: 21 x 145 - ne(sqrt(psi)) on C2 Grid
```

The radial coordinate of the density profile is the square root of `psi`. The `psi` can be found in the `efit` DDA.

```
>> w.efit.psn
ans =
  Shot: 57941
  dda: efit
  uid: JETPPF
n. DT: 3
  psni: 33 x 149 - PSI_NORM (RAD_NORM_IN)
  psnm: 33 x 149 - PSI_NORM(R) ON Z=ZMAG
  psno: 33 x 149 - PSI_NORM (RAD_NORM_OUT)
```

The correct DTYPE is `psnm`. Then we can slice the data at for example 50s. The ECE temperature is also read. The data are also normalized.

```
ne = w.nft2.prof.ct(50) / 1e19;
psi = w.efit.psnm.ct(50);
te = w.ecm1.prfl.ct(50) / 1e3;
```

In order to get the density as function of `R` instead of `psi` it is possible to use the pseudo-field `f` which realizes a sort of functional composition.

```
ner = ne.f(sqrt(psi), 'extreme');
```

Then we plot the `psi`, the density as function of the `psi` and as function of the radius, and the temperature and pressure. Note that the temperature and the density are multiplied together even if they have a different spatial base, and in two plots the radial coordinate is restricted between 2 and 4 with the `:` operator (Fig.1).


```

plot(psi:2:4,' k' )
plot(ne,' k' )
plot(ner:2:4,' k' )
plot(te,' k' ,ner*te,' k-' )

```

CONCLUSION

The use of OO features of MATLAB has greatly simplified the use of this interactive environment. Four classes have been developed, their main methods have been shown together with some examples.

REFERENCES

- [1]. M. Metcalf, J. Reid and M. Cohen “Fortran 95/2003 Explained” Third Edition, Oxford University Press.
- [2]. R. Layne, M. Wheatley “New Data Storage and Retrieval Systems for JET Data” EFDA–JET–PR(01)23

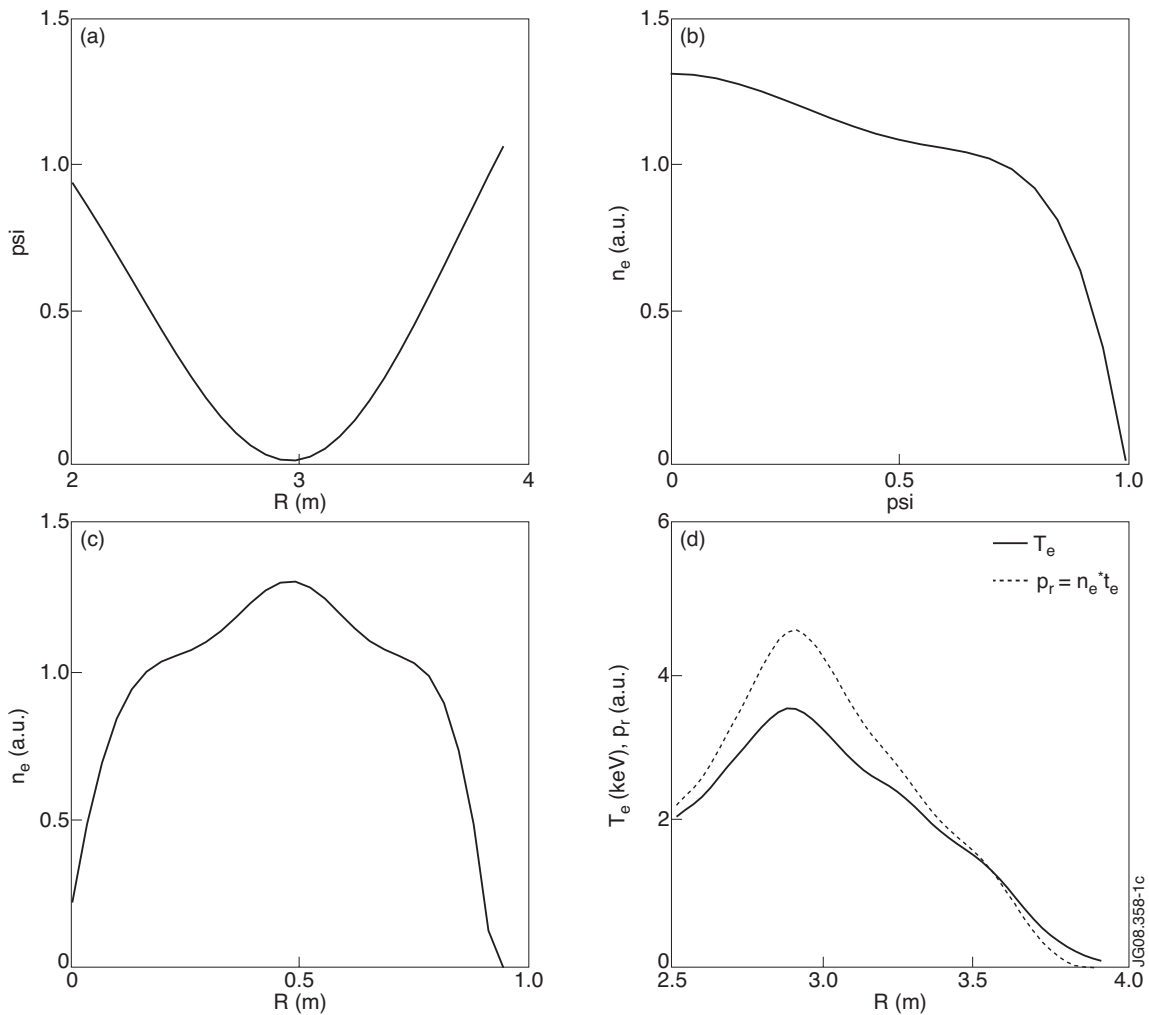


Figure 1: The plots produced by the methods plot as shown in the text. The labels on the x and y axes and the legend have been added later for clarity. (a) the ψ as function of the radius; (b) the density as function of the ψ ; (c) the density as function of the radius obtained combining $n_e(\sqrt{\psi})$ and $\psi(R)$; (d) the temperature and the pressure (in arbitrary units), the latter obtained multiplying temperature and density.