the `o•topus` **developer's**

Chapter 2: xc

f_der: **This is the** f_der_type **structure: the structure that knows how to do the derivatives. It is de ned** inout **for technical reasons, but it should not**

xc_functl

3.1 xc_functl_type

```
integer :: family
integer :: id
integer :: spin_channels
integer(POINTER_SIZE) :: conf
integer(POINTER_SIZE) :: info
```

The integer family **may take one of the four following families:**

XC_FAMILY_LDA = >**or LDA functionals.**

XC_FAMILY_GGA = >**or GGA functionals.**

XC_FAMILY_MGGA 3 >**or MGGA functionals.**

Y_OEP = >**or OEP functionals (i.e. orbital dependent functionals).**

et in either the xc_functl_init_exchange **and** xc_functl_init_
nding on the functional requested, read from the input le, and which
integer identi er.

the identi er for the functional0 (wing)TF**. It ma y tak e the follo wing v alues:**

uncionals:

e XC_FAMILY_LDA **family, any of the exchange functionals listed in Sec-**
LDA functionals], page 11. In fact, only one: XC_LDA_X (wing)TF

ILY_GGA **family, any of the exchange functionals listed in Sec-**
ctionals], page 12.

Y_MGGA **any the exchange functionals listed in Sec-** nput calculationsof spin-unpolarized
2. In fact, only one: XC_MGGA_X_TPSS

ase49.3527 0 Td (=)Tj 9.3490 (wing)T300 (wing)TF)Tj 6.1090817.2473 0 Td (,)Tj 6.94907 0 Td (corresp)Tj 34.36

3.2 subroutine xc_functl_init_exchange

```
type(xc_functl_type), intent(out) :: functl
integer,              intent(in)  :: spin_channels
```

This subroutine initializes one xc_functl_type **structure**

4 lib_xc

This module is in charge of providing the simple functionals, i.e. the LDA, GGA and mGGA functionals. It is in fact nothing else than an interface to the `libxc.a` C library.

The LDA, GGA and mGGA functionals defined here are `local` (yes, the GGA and MGGA are also local), in the sense that the value of the potential at a given point depends only on the values of the density – and the gradient of the density and the kinetic energy density, for the GGA and mGGA cases – at a given point:

$$v_{xc}^{LDA}(\vec{r}) = v_{xc}^{LDA}[n(\vec{r})] \, ;$$

$$v_{xc}^{GGA}(\vec{r}) = v_{xc}^{GGA}[n(\vec{r}); \vec{\nabla} n(\vec{r})] \, ;$$

$$v_{xc}^{mGGA}(\vec{r}) = v_{xc}^{mGGA}[n(\vec{r}); \vec{\nabla} n(\vec{r}); \tau(\vec{r})]$$

Note that these functionals are not the only ones used by the `octopus` code – there is also the possibility of employing

4.2 subroutine xc_lda

```
integer(POINTER_SIZE), intent(in)  :: p
FLOAT,                 intent(in)  ::
```

```
integer,              intent(in)  :: functional
integer,              intent(in)  :: nspin
```

It initializes the p **handler to hold one of the MGGA functionals, which the nspinh**

4.11 GGA functionals

Exchange:

XC_GGA_X_PBE = 101 **=>** **Perdew, Burke & Ernzerhof exchange**

XC_GGA_XC_LB = 103 **=>** **van Leeuwen & Baerends**

Correlation:

XC_GGA_C_PBE = 102 **=>** **Perdew, Burke & Ernzerhof correlation**

4.12 MGGA functionals

Exchange:

XC_MGGA_X_TPSS = 201 **=>** **Perdew, Tao, Staroverov & Scuseria exchange**

Correlation:

XC_MGGA_C_TPSS =