Macros abstracting memory region decorators (`__global`, `__private`, etc.)

```
void NS(DriftExact_track_particle)(
    SIXTRL_PARTICLE_ARGPTR_DEC NS(Particle)* SIXTRL_RESTRICT p,
    SIXTRL_BE_ARGPTR_DEC const NS(DriftExact) *const
        SIXTRL_RESTRICT drift ) SIXTRL_NOEXCEPT
{
    /* Note: Signature & particle type changed
       due to optimisation a) */
    typedef NS(particle_real_type) real_type;
```

Macros abstracting language features

Resolution of `sqrt` like before for C++

```
#if defined( __cplusplus )
using std::sqrt; /* ADL */
#endif /* defined( __cplusplus ) */
```

Reduce the numbers of locally stored expr. relying on `p.px, p.py` being in thread-local memory

```
real_type const length = NS(DriftExact_length)( drift );
real_type const one_plus_delta =
            NS(Particle_delta)( p ) + ( real_type )1.;
real_type lpzi = one_plus_delta * one_plus_delta -
                 NS(Particle_px)( p ) * NS(Particle_px)( p ) -
                 NS(Particle_py)( p ) * NS(Particle_py)( p );
```

Consistency Checks -> **`Disabled / NO-OP`** in Release Mode

```
SIXTRL_ASSERT( p != SIXTRL_NULLPTR );
SIXTRL_ASSERT( drift != SIXTRL_NULLPTR );
SIXTRL_ASSERT( lpzi > ( real_type )0. );

lpzi = length / sqrt( lpzi );
```

Update attributes for p

```
NS(Particle_add_to_x)( p, NS(Particle_px)( p ) * lpzi );
NS(Particle_add_to_y)( p, NS(Particle_py)( p ) * lpzi );
NS(Particle_add_to_s)( p, length );
NS(Particle_add_to_zeta)( p, NS(Particles_rvv)( p ) * length -
                          one_plus_delta * lpzi );

NS(Particle_increment_at_element)( p ); /* Cf. b) */
}
```