

Every mechanism defines: `<mech_name>_reg_(void)`.  
`<mech_name>_reg_(void)` calls `register_mech(mechanism, <mech_name>_alloc, <mech_name>_cur, <mech_name>_jacob, nullptr, nullptr, -1, 1);` and  
`neuron::mechanism::register_data_fields(mechtype, field<double>{"<var_name">"}, field<double>{"<array_var_name">, <Array_size>}, fields...);`  
`register_mech` registers the allocator, `nrn_cur` function, `jacob` function, etc

`register_data_fields`: takes care creating the storage underneath by calling

`neuron::mechanism::detail::register_data_fields(int type, std::vector<std::pair<const char*, int>> const& param_info, std::vector<std::pair<const char*, const char*>> const& dparam_info)`

`neuron::Model::add_mechanism()`

`neuron::container::Mechanism::storage(short mech_type, std::string name, std::vector<Variable> floating_point_fields)`

`floating_point_fields` is a vector of structs that hold the informing for the various RANGE variables of the mechanism

We pass a `std::vector<Variable>` to a wrapper `neuron::container::Mechanism::field::FloatingPoint`. Due to the fact that the wrapper has a `num_instances` non static member function we need to allocate vectors for each one of the RANGE variables in the `m_var_info` member of `FloatingPoint`  
`neuron::container::Mechanism::storage` sets `using base_type = soa<storage, field::FloatingPoint>;`

Creation of `soa` storage struct which holds a tuple of `detail::field_data` structs which holds the underlying storage: `std::tuple<detail::field_data<Tags, detail::has_num_instances_v<Tags>>...> m_data{};`

`struct field_data<Tag, true>{}`