## Validation

There are two steps to validating a hierarchical model. First, the aggregation and synchronization rules are checked to see if they are well-formed, and if the elements they reference exist. Referenced files and models must exist; ‘ports’ must reference portIDs in the relevant namespace of the referenced model; xpaths must point to real elements. These sorts of validation rules are discussed in the main document under the relevant constructs.

Second, if one follows the rules of aggregation and synchronization for each model in the SBML file to create a ‘flattened’ model with no hierarchical structure, the resulting model and modelDefintions must all be valid SBML. The one caveat here is namespace issues: if two submodels containing elements with the same SId or metaID (if they came from different files) are aggregated into the same containing model, they may exist peacefully together in the flattened model.

These validation rules only apply to models defined in the SBML file being validated. Models defined in external files and instantiated as submodels may or may not be valid; the only rule is that the model containing that instantiation must itself be valid. Obviously, malformed XML will be completely unparseable, and the model will be impossible to instantiate at all. But models that leave required flags undefined in its elements may become valid if those elements are replaced by valid elements, or if they are deleted entirely. Similarly, references to nonexistent elements may themselves be deleted, and illegal combinations of objects may be rectified.

The intent behind this rule is that, again, the modeler may not be able to change the model they wish to incorporate, and this gives them a way to create valid SBML while still properly referencing their source.

For translation purposes and to make validation simpler, the following algorithm is proposed to flatten hierarchical models. For every Model object in the file (modelDefinition and model):

1. Examine the submodels of the model you are translating or validating. If any of them themselves contain submodels, repeat this algorithm from the beginning with those submodels before continuing. Afterwards, repeat steps 2-5 for each submodel:
2. Give all submodels that do not yet have one an ID unique to the containing model’s namespace. For example, if the model has three unnamed submodels, one might assign them the names ‘submod1’, ‘submod2’, and ‘submod3’, assuming no other element in the model had those names. Also ensure that no other element ID or metaID \*begins\* with “[modelID]\_”. If it does, add an underscore to the modelID and test again.
3. Remove any submodel elements that have been replaced or deleted.
4. Rename the remaining subelements. Every SId will become ‘[modelID]\_[originalID]’, and every metaID will become ‘[modelID]\_[originalMetaID]’.
5. Rename all SIdRefs and metaIdRefs remaining in the submodel. Every IdRef that pointed to an element that was not removed, replace with ‘[modelID]\_[originalID]’. Every IdRef that pointed to a replaced element, replace with the SId of the replacement element. This is the purpose of ensuring no element in the containing model began with the modelID and an underscore—if they had, we might overlap here, or at least be confusing. If any IdRefs are found that point to a deleted element, stop. The model is invalid, as it contains a reference to a nonexistent element. Note that because containing models may reference sub-subelements directly by using the subelement-child-of-a-subelement construct, the SId or MetaId thus referenced may have been mangled by this algorithm. Be sure to be able to find these new sub-subelement IDs in some way, perhaps by keeping a map of old ID to new IDs.
6. After performing the above tasks for all submodels, assemble all remaining elements of the submodels and of the containing model into a single Model. Merge the various lists (ListOfSpecies, ListOfReactions, etc.), and be sure to include all remaining annotations. If any other packages have defined their own ListOfX elements, ensure that these are merged properly as well.
7. Test the Model for validity.

At this point, the Model is ready to be exported as Level 3 core (plus any attendant packages used by the submodels), or to be declared valid or invalid by the validator. Again, when instantiating a model, one does not have to first test the validity of that model. If it is in the same file as the containing model, it will be tested anyway by this algorithm. If it is in a different file, that file’s validity (or lack thereof) should not affect the validity of the file being tested (though a validator may warn the user of this situation if it desires).