ATS Native XML Input Specification V1

Table of Contents

Notes

- ATS's input spec is similar, but not identical, to Amanzi's Native input spec. Much of this document is stolen from Amanzi's Native Spec.
- Note also that nearly **everything** takes a VerboseObject spec. Assume all specs may take a VerboseObject spec within a sublist named *"VerboseObject"* unless otherwise noted.

Syntax of the Specification

- Input specification for each ParameterList entry consists of two parts. First, a bulleted list defines the usage syntax and available options. This is followed by example snipets of XML code to demonstrate usage.
- In many cases, the input specifies data for a particular parameterized model, and ATS supports a number of parameterizations. For example, initial data might be uniform (the value is required), or linear in y (the value and its gradient are

required). Where ATS supports a number of parameterized models for quantity Z, the available models will be listed by name, and then will be described in the subsequent section. For example, the specification for an *"X"* list might begin with the following:

  - *"Y"* `[string]` **"default_value"**, *"other"*, *"valid"*, *"options"*
  - Z `[Z-spec]` Model for Z, choose exactly one of the following: (1) *"z1"*, or (2) *"z2"* (see below)

Here, an *"X"* is defined by a *"Y"* and a *"Z"*. The *"Y"* is a string parameter but the *"Z"* is given by a model (which will require its own set of parameters). The options for *"Z"* will then be described as a spec:

- *"z1"* applies model z1. Requires *"z1a"* `[string]`
- *"z2"* applies model z2. Requires *"z2a"* `[double]` and *"z2b"* `[int]`

An example of using such a specification:

```
<ParameterList name="X">
  <Parameter name="Y" type="string" value="hello"/>
  <ParameterList name="z2">
    <Parameter name="z2a" type="double" value="0.7"/>
    <Parameter name="z2b" type="int" value="3"/>
  </ParameterList>
</ParameterList>
```

Here, the user is defining X with Y="hello", and Z will be a z2 constructed with z2a=0.7 and z2b=3.

Conventions:

- Reserved keywords and labels are *"quoted and italicized"* -- these labels or values of parameters in user-generated input files must match (using XML matching rules) the specified or allowable values.
- User-defined labels are indicated with ALL-CAPS, and are meant to represent a typical name given by a user - these can be names or numbers or whatever serves best the organization of the user input data.
- Bold values are default values, and are used if the Parameter is not provided.

Main

The main ParameterList frames the entire input spec, and must contain one sublist for each of the following sections. Additionally, for compatibility with Amanzi, the following Parameters are typically present, and should not be changed, as ATS does not currently support Amanzi-S.

- *"Native Unstructured Input"* `[bool]`, Should always be *"true"*
- *"grid_option"* `[string]`, Should always be *"Unstructured"*
- *"Mesh"* `[mesh-spec]` See the Mesh spec.

- *"Domain"* `[domain-spec]` See the Domain spec.
- *"Regions"* `[list]`

  List of multiple Region specs, each in its own sublist named uniquely by the user.
- *"coordinator"* `[coordinator-spec]` See the Coordinator spec.
- *"visualization"* `[visualization-spec]` A Visualization spec for the main mesh/domain.
- *"visualization XX"* `[visualization-spec]`

  Potentially more than one other Visualization specs, one for each domain *"XX"*. e.g. *"surface"*
- *"checkpoint"* `[checkpoint-spec]` A Checkpoint spec.
- *"observations"* `[observation-spec]` An Observation spec.
- *"PKs"* `[list]`

  A list containing exactly one sublist, a PK spec with the top level PK.
- *"state"* `[list]` A State spec.

Mesh

The mesh represents the primary domain of simulation. Simple, structured meshes may be generated on the fly, or complex unstructured meshes are provided as `Exodus II` files.

Additionally, several other domains may be constructed from the main domain specified as the mesh. These include surface meshes, which are ripped from the sub-surface mesh.

Example of a mesh generated internally:

```
<ParameterList name="Mesh">
  <ParameterList name="Unstructured"/>
    <ParameterList name="Generate Mesh"/>
      <ParameterList name="Uniform Structured"/>
        <Parameter name="Number of Cells" type="Array(int)" value="{100, 1, 100
        <Parameter name="Domain Low Coordinate" type="Array(double)" value="{0.
        <Parameter name="Domain High Coordinate" type="Array(double)" value="{1
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Example of a mesh read from an external file, along with a surface mesh:

```
<ParameterList name="Mesh">
  <Parameter name="Framework" type="string" value="MSTK"/>
  <ParameterList name="Read Mesh File">
```

```
    <Parameter name="File" type="string" value="mesh_filename.exo"/>
    <Parameter name="Format" type="string" value="Exodus II"/>
  </ParameterList>
  <ParameterList name="Surface Mesh">
     <Parameter  name="surface sideset name" type="string" value="surface_regio
  </ParameterList>
</ParameterList>
```

Note that in this case, ATS expects there to also be a Region spec (in this example named) *"surface_region"* which describes a face set of the main mesh.

Domain

The domain simply refers to the geometric model in which a mesh is contained. Currently it has a single parameter.

- *"Spatial Dimension"* `[int]` **3**

Region

Regions are geometrical constructs used in ATS to define subsets of the computational domain in order to specify the problem to be solved, and the output desired. Regions may represents zero-, one-, two- or three-dimensional subsets of physical space. For a three-dimensional problem, the simulation domain will be a three-dimensional region bounded by a set of two-dimensional regions. If the simulation domain is N-dimensional, the boundary conditions must be specified over a set of regions are (N-1)-dimensional.

User-defined regions are constructed using the following syntax

- REGION-SHAPE [list] In this case the list name is a geometric model primitive, from the table below.

shape functional name	**parameters**	**type(s)**	**Comment**
"Region: Point" [SU]	*"Coordinate"*	Array(double)	Location of point in space
"Region: Box" [SU]	*"Low Coordinate"*, *"High Coordinate"*	Array(double), Array(double)	Location of boundary points of box
"Region: Plane" [SU]	*"Direction"*, *"Location"*	string, double	direction: *"X"*, *"-X"*, etc, and *"Location"* is coordinate value
"Region: Polygon" [U]	*"Number of points"*, *"Points"*	int, Array double	Number of polygon points and point coordinates in linear array

... continued on next page

shape functional name	**parameters**	**type(s)**	**Comment**
"Region: Labeled Set"	*"Label"*, *"File"*, *"Format"*, *"Entity"*	string, string, string, string	**Set per label defined in mesh file (see below)** (available for frameworks supporting the *"File"* keyword)
"Region: Color Function" [S]	*"File"*, *"Value"*	string, int	Set defined by color in a tabulated function file (see below)
"Region: Layer"	*"File#"*, *"Label#"*	(#=1,2) string, string	Region between two surfaces
"Region: Surface"	*"File"* *"Label"*	string, string	Labeled triangulated face set in file

Notes

- *"Region: Point"* defines a point in space. Using this definition, cell sets encompassing this point are retrieved inside ATS.
- *"Region: Box"* defines a region bounded by coordinate-aligned planes. Boxes are allowed to be of zero thickness in only one direction in which case they are equivalent to planes.
- Currently, *"Region: Plane"* is constrained to be coordinate-aligned.
- The *"Region: Labeled Set"* region defines a named set of mesh entities existing in an input mesh file. This is the same file that contains the computational mesh. The name of the entity set is given by *"Label"*. For example, a mesh file in the Exodus II format can be processed to tag cells, faces and/or nodes with specific labels, using a variety of external tools. Regions based on such sets are assigned a user-defined label for ATS, which may or may not correspond to the original label in the exodus file. Note that the file used to express this labeled set may be in any ATS-supported mesh format (the mesh format is specified in the parameters for this option). The *"entity"* parameter may be necessary to specify a unique set. For example, an Exodus file requires *"Cell"*, *"Face"* or *"Node"* as well as a label (which is an integer). The resulting region will have the dimensionality associated with the entities in the indicated set.

  By definition, "Labeled Set" region is applicable only to the unstructured version of ATS.

  Currently, ATS only supports mesh files in the Exodus II format.
- *"Region: Color Function"* defines a region based a specified integer color, *"Value"*, in a structured color function file, *"File"*. The format of the color function file is given below in the "Tabulated function file format" section. As shown in the file, the color values may be specified at the nodes or cells of the color function grid. A computational cell is assigned the 'color' of the data grid cell containing its cell centroid (cell-based colors) or the data grid nearest its cell-centroid

(node-based colors). Computational cells sets are then built from all cells with the specified color *"Value"*.

In order to avoid, gaps and overlaps in specifying materials, it is strongly recommended that regions be defined using a single color function file.

- *"Region: Polygon"* defines a polygonal region on which mesh faces and nodes can be queried. NOTE that one cannot ask for cells in a polygonal region.In 2D, the "polygonal" region is a line and is specified by 2 points In 3D, the "polygonal" region is specified by an arbitrary number of points. In both cases the point coordinates are given as a linear array. The polygon can be non-convex.

  The polygonal region can be queried for a normal. In 2D, the normal is defined as [Vy,-Vx] where [Vx,Vy] is the vector from point 1 to point 2. In 3D, the normal of the polygon is defined by the order in which points are specified.

- Surface files contain labeled triangulated face sets. The user is responsible for ensuring that the intersections with other surfaces in the problem, including the boundaries, are *"exact"* (*i.e.* that surface intersections are *"watertight"* where applicable), and that the surfaces are contained within the computational domain. If nodes in the surface fall outside the domain, the elements they define are ignored.

  Examples of surface files are given in the *"Exodus II"* file format here.

- Region names must NOT be repeated

Example:

```
<ParameterList name="Regions">
  <ParameterList name="Top Section">
    <ParameterList name="Region: Box">
      <Parameter name="Low Coordinate" type="Array(double)" value="{2, 3, 5}"/>
      <Parameter name="High Coordinate" type="Array(double)" value="{4, 5, 8}"/
    </ParameterList>
  </ParameterList>
  <ParameterList name="Middle Section">
    <ParameterList name="Region: Box">
      <Parameter name="Low Coordinate" type="Array(double)" value="{2, 3, 3}"/>
      <Parameter name="High Coordinate" type="Array(double)" value="{4, 5, 5}"/
    </ParameterList>
  </ParameterList>
  <ParameterList name="Bottom Section">
    <ParameterList name="Region: Box">
      <Parameter name="Low Coordinate" type="Array(double)" value="{2, 3, 0}"/>
      <Parameter name="High Coordinate" type="Array(double)" value="{4, 5, 3}"/
    </ParameterList>
  </ParameterList>
  <ParameterList name="Inflow Surface">
    <ParameterList name="Region: Labeled Set">
      <Parameter name="Label"  type="string" value="sideset_2"/>
      <Parameter name="File"   type="string" value="F_area_mesh.exo"/>
      <Parameter name="Format" type="string" value="Exodus II"/>
```

```
      <Parameter name="Entity" type="string" value="Face"/>
    </ParameterList>
  </ParameterList>
  <ParameterList name="Outflow plane">
    <ParameterList name="Region: Plane">
      <Parameter name="Location" type="Array(double)" value="{0.5, 0.5, 0.5}"/>
      <Parameter name="Direction" type="Array(double)" value="{0, 0, 1}"/>
    </ParameterList>
  </ParameterList>
  <ParameterList name="Sand">
    <ParameterList name="Region: Color Function">
      <Parameter name="File" type="string" value="F_area_col.txt"/>
      <Parameter name="Value" type="int" value="25"/>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

In this example, "Top Section", "Middle Section" and "Bottom Section" are three box-shaped volumetric regions. "Inflow Surface" is a surface region defined in an Exodus II-formatted labeled set file and "Outflow plane" is a planar region. "Sand" is a volumetric region defined by the value 25 in color function file.

Coordinator

In the *"coordinator"* sublist the user specifies global control of the simulation, including starting and ending times and restart options.

- *"start time"* `[double]`, **0.**
- *"start time units"* `[string]`, **"s"**, *"d"*, *"yr"*
- *"end time"* `[double]`
- *"end time units"* `[string]`, **"s"**, *"d"*, *"yr"*
- *"end cycle"* `[int]`
- *"restart from checkpoint file"* `[string]`

  requires a path to the checkpoint file.
- *"wallclock end time"* [double]

  ?? This works, but this documentation needs updated.
- *"required times"* `[time-control-spec]`

  A TimeControl spec that sets a collection of times/cycles at which the simulation is guaranteed to hit exactly. This is useful for situations such as where data is provided at a regular interval, and interpolation error related to that data is to be minimized.

Note that either *"end cycle"* or *"end time"* are required, and if both are present, the simulation will stop with whichever arrives first. An *"end cycle"* is commonly used to ensure that, in the case of a time step crash, we do not continue on forever spewing output.

Example:

Visualization

A user may request periodic writes of field data for the purposes of visualization in the *"visualization"* sublists. ATS accepts a visualization list for each domain/mesh -- currently this is up to two (one for the subsurface, and one for the surface). These are in separate ParameterLists, entitled *"visualization"* for the main mesh, and *"visualization surface"* on the surface mesh. It is expected that, for any addition meshes, each will have a domain name and therefore admit a spec of the form: *"visualization DOMAIN-NAME"*.

Each list contains all parameters as in a TimeControl spec, and also:

- *"file name base"* `[string]` **"visdump_data", "visdump_surface_data"**
- *"dynamic mesh"* `[bool]` **false**

  Write mesh data for every visualization dump, this facilitates visualizing deforming meshes.

Currently not supported...

- *"regions"* `[Array(string)]` **empty array**

  Write an array into the visualization file that can be used to identify a region or regions. The first entry in the regions array is marked with the value 1.0 in the array, the second with the value 2.0, and so forth. The code ignores entries in the regions array that are not valid regions that contain cells.

- *"write partition"* `[bool]` **false**

  If this parameter is true, then write an array into the visualization file that contains the rank number of the processor that owns a mesh cell.

Example:

```
<ParameterList name="visualization">
  <Parameter name="file name base" type="string" value="visdump_data"/>

  <Parameter name="cycles start period stop" type="Array(int)" value="{0, 100,
  <Parameter name="cycles" type="Array(int)" value="{999, 1001}" />

  <Parameter name="times start period stop 0" type="Array(double)" value="{0.0,
  <Parameter name="times start period stop 1" type="Array(double)" value="{100.
  <Parameter name="times" type="Array(double)" value="{101.0, 303.0, 422.0}"/>

  <Parameter name="dynamic mesh" type="bool" value="false"/>
</ParameterList>
```

Checkpoint

A user may request periodic dumps of ATS Checkpoint Data in the *"checkpoint"* sublist. The user has no explicit control over the content of these files, but has the guarantee that the ATS run will be reproducible (with accuracies determined by machine round errors and randomness due to execution in a parallel computing environment). Therefore, output controls for Checkpoint Data are limited to file name generation and

writing frequency, by numerical cycle number. Unlike *"visualization"*, there is only one *"checkpoint"* list for all domains/meshes.

The checkpoint-spec includes all parameters as in a TimeControl spec and additionally:

- *"file name base"* `[string]` **"checkpoint"**
- *"file name digits"* `[int]` **5**

Example:

```
<ParameterList name="checkpoint">
  <Parameter name="cycles start period stop" type="Array(int)" value="{0, 100,
  <Parameter name="cycles" type="Array(int)" value="{999, 1001}" />
  <Parameter name="times start period stop 0" type="Array(double)" value="{0.0,
  <Parameter name="times start period stop 1" type="Array(double)" value="{100.
  <Parameter name="times" type="Array(double)" value="{101.0, 303.0, 422.0}"/>
</ParameterList>
```

In this example, checkpoint files are written when the cycle number is a multiple of 100, every 10 seconds for the first 100 seconds, and every 25 seconds thereafter, along with times 101, 303, and 422. Files will be written in the form: *"checkpoint00000.h5"*.

Observation

This is not currently correct!

A user may request any number of specific observations from ATS. Each labeled Observation Data quantity involves a field quantity, a model, a region from which it will extract its source data, and a list of discrete times for its evaluation. The observations are evaluated during the simulation and returned to the calling process through one of ATS arguments.

- *"Observation Data"* [list] can accept multiple lists for named observations (OBSERVATION)
    - *"Observation Output Filename"* [string] user-defined name for the file that the observations are written to.
    - OBSERVATION [list] user-defined label, can accept values for *"Variables"*, *"Functional"*, *"Region"*, and all TimeControl spec options.
        - *"Variables"* [Array(string)] a list of field quantities taken from the list of available field quantities:
            - Volumetric water content [volume water / bulk volume]
            - Aqueous saturation [volume water / volume pore space]
            - Aqueous pressure [Pa]
            - Hydraulic Head [m]

    - XXX Aqueous concentration [moles of solute XXX / volume water in MKS] (name formed by string concatenation, given the definitions in *"Phase Definition"* section)
    - X-, Y-, Z- Aqueous volumetric fluxe [m/s]
    - MaterialID
  - *"Functional"* [string] the label of a function to apply to each of the variables in the variable list (Function options detailed below)
  - *"Region"* [string] the label of a user-defined region

The following Observation Data functionals are currently supported. All of them operate on the variables identified.

- *"Observation Data: Point"* returns the value of the field quantity at a point
- *"Observation Data: Integral"* returns the integral of the field quantity over the region specified

Example:

```
<ParameterList name="Observation Data">
  <Parameter name="Observation Output Filename" type="string" value="obs_output
  <ParameterList name="some observation name">
    <Parameter name="Region" type="string" value="some point region name"/>
    <Parameter name="Functional" type="string" value="Observation Data: Point"/
    <Parameter name="Variable" type="string" value="Volumetric water content"/>
    <Parameter name="times" type="Array(double)" value="{100000.0, 200000.0}"/>

    <Parameter name="cycles" type="Array(int)" value="{100000, 200000, 400000,
    <Parameter name="cycles start period stop" type="Array(int)" value="{0, 100

    <Parameter name="times start period stop 0" type="Array(double)" value="{0.
    <Parameter name="times start period stop 1" type="Array(double)" value="{10
    <Parameter name="times" type="Array(double)" value="{101.0, 303.0, 422.0}"/

  </ParameterList>
</ParameterList>
```

PK

The *"PKs"* ParameterList in Main is expected to have one and only one sublist, which corresponds to the PK at the top of the PK tree. This top level PK is also often an MPC (MPCs are PKs).

All PKs have the following parameters in their spec:

- *"PK type"* `[string]`

  The PK type is a special key-word which corresponds to a given class in the PK factory. See available PK types listed below in the Physical PKs section.

- *"PK name"* `[string]` **LIST-NAME**

  This is automatically written as the *"name"* attribute of the containing PK sublist, and need not be included by the user.

Example:

```
<ParameterList name="PKs">
  <ParameterList name="my cool PK">
    <Parameter name="PK type" type="string" value="my cool PK"/>
     ...
  </ParameterList>
</ParameterList>

<ParameterList name="PKs">
  <ParameterList name="Top level MPC">
    <Parameter name="PK type" type="string" value="strong MPC"/>
     ...
  </ParameterList>
</ParameterList>
```

Each PK, which may be named arbitrarily, is one of the following PK specs listed below.

Base PKs

There are several types of PKs, and each PK has its own valid input spec. However, there are three main types of PKs, from which nearly all PKs derive. Note that none of these are true PKs and cannot stand alone.

PKDefaultBase

`PKDefaultBase` is not a true PK, but is a helper for providing some basic functionality shared by (nearly) all PKs. Therefore, (nearly) all PKs inherit from this base class. No input required.

PKPhysicalBase

`PKPhysicalBase (v)-->` PKDefaultBase

`PKPhysicalBase` is a base class providing some functionality for PKs which are defined on a single mesh, and represent a single process model. Typically all leaves of the PK tree will inherit from `PKPhysicalBase`.

- *"domain"* `[string]` **""**, e.g. *"surface"*.

  Domains and meshes are 1-to-1, and the empty string refers to the main domain or mesh. PKs defined on other domains must specify which domain/mesh they refer to.

- *"primary variable key"* `[string]`

  The primary variable associated with this PK, i.e. *"pressure"*, *"temperature"*, *"surface_pressure"*, etc.

- *"initial condition"* `[initial-condition-spec]` See InitialConditions.

  Additionally, the following parameters are supported:

  - *"initialize faces from cell"* `[bool]` **false**

    Indicates that the primary variable field has both CELL and FACE objects, and the FACE values are calculated as the average of the neighboring cells.
  - other, PK-specific additions

PKBDFBase

`PKBDFBase (v)` --> PKDefaultBase

`PKBDFBase` is a base class from which PKs that want to use the `BDF` series of implicit time integrators must derive. It specifies both the `BDFFnBase` interface and implements some basic functionality for `BDF` PKs.

- *"initial time step"* `[double]` **1.**

  The initial timestep size for the PK, this ensures that the initial timestep will not be **larger** than this value.

- *"assemble preconditioner"* `[bool]` **true**

  A flag for the PK to not assemble its preconditioner if it is not needed by a controlling PK. This is usually set by the MPC, not by the user.

In the top-most (in the PK tree) PK that is meant to be integrated implicitly, several additional specs are included. For instance, in a strongly coupled flow and energy problem, these specs are included in the `StrongMPC` that couples the flow and energy PKs, not to the flow or energy PK itself.

- *"time integrator"* `[time-integrator-spec]` is a TimeIntegrator.

  Note that this is only provided in the top-most `PKBDFBase` in the tree -- this is often a StrongMPC or a class deriving from StrongMPC, not a PKPhysicalBDFBase.

- *"preconditioner"* `[preconditioner-spec]` is a Preconditioner.

  This spec describes how to form the (approximate) inverse of the preconditioner.

PKPhysicalBDFBase

`PKPhysicalBDFBase` --> PKBDFBase `PKPhysicalBDFBase` --> PKPhysicalBase `PKPhysicalBDFBase (v)` --> PKDefaultBase

A base class for all PKs that are all of the above.

- *"debug cells"* [Array(int)]

  List of global cell IDs for which (if the verbosity is set high enough) more debugging info is printed to the log file.

- *"absolute error tolerance"* [double] **1.0**

  Absolute tolerance, a_tol in the equation below.

- *"relative error tolerance"* [double] **1.0**

  Relative tolerance, r_tol in the equation below.

By default, the error norm used by solvers is given by: $ENORM(u,du) = |du|/(a_tol + r_tol * |u|)$

Physical PKs

Physical PKs are the physical capability implemented within ATS.

Flow PKs

Richards PK

Permafrost Flow PK

Overland Flow, head primary variable PK

Overland Flow, pressure primary variable, PK

Snow Distribution PK

Energy PKs

Advection Diffusion PK

Energy Base PK

Two-Phase subsurface Energy PK

Three-Phase subsurface Energy PK

Three-Phase subsurface Energy PK

Surface Ice Energy PK

Surface Energy Balance PKs

Surface Energy Balance / Snow -- Monolithic Version

Surface Energy Balance -- Generic Version

Biogeochemistry

Biogeochemistry -- Monolithic Version

Deformation

Volumetric Deformation

MPCs

MPCs couple other PKs, and are the non-leaf nodes in the PK tree.

WeakMPC

StrongMPC

Physical MPCs

Often coupling is an art, and requires special off-diagonal work. Physical MPCs can derive from default MPCs to provide special work.

Coupled Water MPC

Subsurface MPC

Permafrost MPC

State

State consists of two sublists, one for evaluators and the other for atomic constants. The latter is currently called *"initial conditions"*, which is a terrible name which must be fixed.

example:

```
<ParameterList name="state">
  <ParameterList name="field evaluators">
    ...
  </ParameterList>
  <ParameterList name="initial conditions">
    ...
  </ParameterList>
</ParameterList>
```

Field Evaluators

Many field evaluators exist, but most derive from one of four base types.

Field Evaluator Base Classes

PrimaryVariableEvaluator

SecondaryVariableEvaluator

SecondaryVariablesEvaluator

IndependentVariableEvaluator While these provide base functionality, all of the physics are in the following

InitialConditions

Initial condition specs are used in two places -- in the PK spec which describes the initial condition of primary variables, and in the initial conditions sublist of state, in which the value of atomic constants are provided. In Amanzi, this list is also used for initial conditions of primary variables are specified here, not within the PK list (hence the name of this sublist). In ATS, this sublist is pretty much only used for constant scalars and constant vectors.

This list needs to be renamed -- it has nothing to do with inital conditions anymore.

Initialization of constant scalars

A constant scalar field is the global (with respect to the mesh) constant. At the moment, the set of such fields includes atmospheric pressure. The initialization requires to provide a named sublist with a single parameter *"value"*.

```
<ParameterList name="fluid_density">
  <Parameter name="value" type="double" value="998.0"/>
</ParameterList>
```

Initialization of constant vectors

A constant vector field is the global (with respect to the mesh) vector constant. At the moment, the set of such vector constants includes gravity. The initialization requires to provide a named sublist with a single parameter *"Array(double)"*. In two dimensions, is looks like

```
<ParameterList name="gravity">
  <Parameter name="value" type="Array(double)" value="{0.0, -9.81}"/>
</ParameterList>
```

Initialization of scalar fields

A variable scalar field is defined by a few functions (labeled for instance, *"Mesh Block i"* with non-overlapping ranges. The required parameters for each function are *"region"*, *"component"*, and the function itself.

```
<ParameterList name="porosity">
  <ParameterList name="function">
    <ParameterList name="Mesh Block 1">
      <Parameter name="region" type="string" value="Computational domain"/>
      <Parameter name="component" type="string" value="cell"/>
      <ParameterList name="function">
        <ParameterList name="function-constant">
          <Parameter name="value" type="double" value="0.2"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
    <ParameterList name="Mesh Block 2">
      ...
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Initialization of tensor fields

A variable tensor (or vector) field is defined similarly to a variable scalar field. The difference lies in the definition of the function which is now a multi-values function. The required parameters are *"Number of DoFs"* and *"Function type"*.

```
<ParameterList name="function">
  <Parameter name="Number of DoFs" type="int" value="2"/>
  <Parameter name="Function type" type="string" value="composite function"/>
  <ParameterList name="DoF 1 Function">
    <ParameterList name="function-constant">
      <Parameter name="value" type="double" value="1.9976e-12"/>
    </ParameterList>
  </ParameterList>
  <ParameterList name="DoF 2 Function">
    <ParameterList name="function-constant">
      <Parameter name="value" type="double" value="1.9976e-13"/>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Initialization from a file

Some data can be initialized from files. Additional sublist has to be added to named sublist of the *"state"* list with the file name and the name of attribute. For a serial run, the file extension must be *".exo"*. For a parallel run, it must be *".par"*. Here is an example:

```
<ParameterList name="permeability">
  <ParameterList name="exodus file initialization">
    <Parameter name="file" type="string" value="mesh_with_data.exo"/>
    <Parameter name="attribute" type="string" value="perm"/>
  </ParameterList>
</ParameterList>
```

example:

```
<ParameterList name="state">
  <ParameterList name="initial conditions">
    <ParameterList name="fluid_density">
      <Parameter name="value" type="double" value="998.0"/>
    </ParameterList>

    <ParameterList name="fluid_viscosity">
      <Parameter name="value" type="double" value="0.001"/>
    </ParameterList>

    <ParameterList name="gravity">
      <Parameter name="value" type="Array(double)" value="{0.0, -9.81}"/>
    </ParameterList>

  </ParameterList>
</ParameterList>
```

Time integrators, solvers, and other mathematical specs

Common specs for all solvers and time integrators, used in PKs.

TimeIntegrator

Linear Solver Spec

For each solver, a few parameters are used:

- *"iterative method"* `[string]` *"pcg"*, *"gmres"*, or *"nka"*
  defines which method to use.
- *"error tolerance"* `[double]` **1.e-6** is used in the convergence test.
- *"maximum number of iterations"* `[int]` **100** is used in the convergence test.
- *"convergence criteria"* `[Array(string)]` **{"relative rhs"}** specifies multiple convergence criteria. The list may include *"relative residual"*, *"relative rhs"*, and *"absolute residual"*, and *"???? force once????"*
- *"size of Krylov space"* `[int]` is used in GMRES iterative method. The default value is 10.

```
<ParameterList name="my solver">
  <Parameter name="iterative method" type="string" value="gmres"/>
  <Parameter name="error tolerance" type="double" value="1e-12"/>
  <Parameter name="maximum number of iterations" type="int" value="400"/>
  <Parameter name="convergence criteria" type="Array(string)" value="{relative
  <Parameter name="size of Krylov space" type="int" value="10"/>

  <ParameterList name="VerboseObject">
    <Parameter name="Verbosity Level" type="string" value="high"/>
  </ParameterList>
</ParameterList>
```

Preconditioner

These can be used by a process kernel lists to define a preconditioner. The only common parameter required by all lists is the type:

- *"preconditioner type"* `[string]` **"identity"**, *"boomer amg"*, *"trilinos ml"*, *"block ilu"* ???
- *"PC TYPE parameters"* `[list]` includes a list of parameters specific to the type of PC.

Example:

```
<ParameterList name="my preconditioner">
  <Parameter name="type" type="string" value="trilinos ml"/>
   <ParameterList name="trilinos ml parameters"> ?????? check me!
       ...
   </ParameterList>
</ParameterList>
```

Hypre AMG

Internal parameters of Boomer AMG includes

- *"tolerance"* `[double]` **0.0** if is not zero, the preconditioner is dynamic and approximate the inverse matrix with the prescribed tolerance (in the energy norm?).
- *"smoother sweeps"* `[int]` Number of smoothing iterations at each level on each cycle.
- *"cycle applications"* `[int]` **5** Number of V/W cycles to take.
- *"strong threshold"* `[double]` **0.5** Tolerance for including an off-diagonal when coarsening? This is a very tunable parameter.
- *"relaxation type"* `[int]` **6** defines the smoother to be used. Default is 6 which specifies a symmetric hybrid Gauss-Seidel / Jacobi hybrid method.

- *"verbosity"* `[int]` **0** prints BoomerAMG statistics useful for analysis.

- *"number of functions"* `[int]` **1** Used in systems, this is very important to set correctly if you have two or more seperate variables interleaved.

```
<ParameterList name="boomer amg parameters">
  <Parameter name="tolerance" type="double" value="0.0"/>
  <Parameter name="smoother sweeps" type="int" value="3"/>
  <Parameter name="cycle applications" type="int" value="5"/>
  <Parameter name="strong threshold" type="double" value="0.5"/>
  <Parameter name="relaxation type" type="int" value="6"/>
  <Parameter name="verbosity" type="int" value="0"/>
  <Parameter name="number of functions" type="int" value="1"/>
</ParameterList>
```

Trilinos ML

Internal parameters of Trilinos ML includes

```
<ParameterList name="ml parameters">
  <Parameter name="ML output" type="int" value="0"/>
  <Parameter name="aggregation: damping factor" type="double" value="1.33"/>
  <Parameter name="aggregation: nodes per aggregate" type="int" value="3"/>
  <Parameter name="aggregation: threshold" type="double" value="0.0"/>
  <Parameter name="aggregation: type" type="string" value="Uncoupled"/>
  <Parameter name="coarse: type" type="string" value="Amesos-KLU"/>
  <Parameter name="coarse: max size" type="int" value="128"/>
  <Parameter name="coarse: damping factor" type="double" value="1.0"/>
  <Parameter name="cycle applications" type="int" value="2"/>
  <Parameter name="eigen-analysis: iterations" type="int" value="10"/>
  <Parameter name="eigen-analysis: type" type="string" value="cg"/>
  <Parameter name="max levels" type="int" value="40"/>
  <Parameter name="prec type" type="string" value="MGW"/>
```

```
  <Parameter name="smoother: damping factor" type="double" value="1.0"/>
  <Parameter name="smoother: pre or post" type="string" value="both"/>
  <Parameter name="smoother: sweeps" type="int" value="2"/>
  <Parameter name="smoother: type" type="string" value="Gauss-Seidel"/>
</ParameterList>
```

Block ILU

The internal parameters of the block ILU are as follows:

```
<ParameterList name="block ilu parameters">
  <Parameter name="fact: relax value" type="double" value="1.0000000000000000e
  <Parameter name="fact: absolute threshold" type="double" value="0.0000000000
  <Parameter name="fact: relative threshold" type="double" value="1.0000000000
  <Parameter name="fact: level-of-fill" type="int" value="0"/>
  <Parameter name="overlap" type="int" value="0"/>
  <Parameter name="schwarz: combine mode" type="string" value="Add"/>
</ParameterList>
```

Indentity

The default, no PC applied.

NonlinearSolver

Other Common Specs

TimeControl

The time-control-spec is used for multiple lists that need to indicate simulation times or cycles on which to do something.

- *"cycles start period stop"* `[Array(int)]`
  The first entry is the start cycle, the second is the cycle period, and the third is the stop cycle or -1, in which case there is no stop cycle. A visualization dump is written at such cycles that satisfy cycle = start + n*period, for n=0,1,2,... and cycle < stop if stop != -1.0.
- *"cycles start period stop N"* `[Array(int)]`
  If multiple cycles start period stop parameters are needed, then use these parameters with N=0,1,2,...
- *"cycles"* `[Array(int)]`
  An array of discrete cycles that at which a visualization dump is written.
- *"times start period stop"* `[Array(double)]`
  The first entry is the start time, the second is the time period, and the third is the stop time or -1, in which case there is no stop time. A visualization dump is written at such times that satisfy time = start + n*period, for n=0,1,2,... and time < stop if stop != -1.0. Note that all times units are in seconds.

- *"times start period stop n"* `[Array(double)]`
  If multiple start period stop parameters are needed, then use this these parameters with n=0,1,2,..., and not the single *"times start period stop"* parameter. Note that all times units are in seconds.
- *"times"* `[Array(double)]`
  An array of discrete times that at which a visualization dump shall be written. Note that all times units are in seconds.

VerboseObject

`Teuchos::VerboseObject` is a tool for managing code output. See also the Trilinos documentation

- *"Verbosity Level"* `[string]` **GLOBAL_VERBOSITY**, *"low"*, *"medium"*, *"high"*, *"extreme"* The default is set by the global verbosity spec, (fix me!) Typically, *"low"* prints out minimal information, *"medium"* prints out errors and overall high level information, *"high"* prints out basic debugging, and *"extreme"* prints out local debugging information. *"medium"* is the standard.

Function

To set up non-trivial boundary conditions and/or initial fields, ATS supports a few mathematical functions. New function types can added easily. Each function is defined a list:

```
<ParameterList name="NAME">
  function-specification
</ParameterList>
```

The parameter list name string *"NAME"* is arbitrary and meaningful only to the parent parameter list. This list is given as input to the ATS::FunctionFactory::Create method which instantiates a new ATS::Function object. The function-specification is one of the following parameter lists.

????? many more are supported... add to the list! ????

Constant function

Constant function is defined as *f(x) = a*, for all *x*. The specification of this function needs only one parameter. For example, when *a = 1*, we have:

```
<ParameterList name="function-constant">
  <Parameter name="value" type="double" value="1.0"/>
</ParameterList>
```

Tabular function

Given values $x_i, y_i, i = 0, \ldots n-1$, a tabular function $f(x)$ is defined piecewise:

$$\begin{aligned} f(x) &= x_0, & x \le x_0, \\ f(x) &= f(x_{i-1}) + (x - x_{i-1}) \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}, & x \in (x_{i-1}, x_i], \\ f(x) &= x_{n-1}, & x > x_{n-1}. \end{aligned}$$

This function is continuous and linear between two consecutive points. This behavior can be changed using parameter *"forms"*. This parameter is optional. If specified it must be an array of length equal to one less than the length of *x values*. Each value is either *"linear"* to indicate linear interpolation on that interval, or *"constant"* to use the left endpoint value for that interval. The example defines function that is zero on interval $(-\infty, 0]$, linear on interval $(0, 1]$, constant (*f(x)=1*) on interval $(1, 2]$, and constant (*f(x)=2*) on interval $(2, \infty]$.

```
<ParameterList name="function-tabular">
  <Parameter name="x values" type="Array(double)" value="{0.0, 1.0, 2.0}"/>
  <Parameter name="y values" type="Array(double)" value="{0.0, 1.0, 2.0}"/>
  <Parameter name="forms" type="Array(string)" value="{linear, constant}"/>
</ParameterList>
```

Smooth step function

A smooth C^2 function *f(x)* on interval $[x_0, x_1]$ is defined such that *f(x) = y_0* for *x < x0*, *f(x) = y_1* for *x > x_1*, and monotonically increasing for $x \in [x_0, x_1]$. Here is an example:

```
<ParameterList name="function-smooth-step">
  <Parameter name="x0" type="double" value="0.0"/>
  <Parameter name="y0" type="double" value="0.0"/>
  <Parameter name="x1" type="double" value="1.0"/>
  <Parameter name="y1" type="double" value="2.0"/>
</ParameterList>
```

Polynomial function

A generic polynomial function is given by the following expression:

$$f(x) = \sum_{j=0}^{n} c_j (x - x_0)^{p_j}$$

where c_j are coefficients of monomials, p_j are integer exponents, and x_0 is the reference point. Here i san example of a quartic polynomial:

```
<ParameterList name="function-polynomial">
  <Parameter name="coefficients" type="Array(double)" value="{1.0, 1.0}"/>
  <Parameter name="exponents" type="Array(int)" value="{2, 4}"/>
  <Parameter name="reference point" type="double" value="0.0"/>
</ParameterList>
```

Multi-variable linear function

A multi-variable linear function is formally defined by

$$f(x) = y_0 + \sum_{j=0}^{n-1} g_j (x_j - x_{0,j})$$

with the constant term "math:*y_0* and gradient $g_0, g_1 \ldots, g_{n-1}$. If the reference point x_0 is specified, it must have the same number of values as the gradient. Otherwise, it

defaults to zero. Note that one of the parameters in a multi-valued linear function can be time. Here is an example:

```
<ParameterList name="function-linear">
  <Parameter name="y0" type="double" value="1.0"/>
  <Parameter name="gradient" type="Array(double)" value="{1.0, 2.0, 3.0}"/>
  <Parameter name="x0" type="Array(double)" value="{2.0, 3.0, 1.0}"/>
</ParameterList>
```

Separable function

A separable function is defined as the product of other functions such as

$$f(x_0, x_1, ..., x_{n-1}) = f_1(x_0)\, f_2(x_1, ..., x_{n-1})$$

where f_1 is defined by the *"function1"* sublist, and f_2 by the *"function2"* sublist:

```
<ParameterList name="function-separable">
  <ParameterList name="function1">
    function-specification
  </ParameterList>
  <ParameterList name="function2">
    function-specification
  </ParameterList>
</ParameterList>
```