MPAS Streams: Requirements and Design

Introduction

The long-term objective of this development is to implement the ability to define an arbitrary set of I/O *streams* in MPAS at run-time. For the purposes of this document, a *stream* refers to a collection of fields that are read or written from the same file at the same interval; a stream is uniquely defined by:

1. A specification for how filenames are laid out in the file system, including:
    a. The filename a stream should point to, or a template for the filename, including a potential directory structure.
    b. A reference time, specifying the first timestamp that will occur in the series of files.
    c. An interval between files, specifying how often filename breaks occur.
2. A specification of when the stream is to be read or written.
3. A unique identifier associated with the stream, which will be used to refer to the stream in the MPAS code.
4. Whether the stream is an input stream, an output stream, or both an input and an output stream.
5. A list of fields to be included in the stream, i.e., a list of fields to be read when the stream is read, or written when the stream is written.
6. The precision of the stream, and all fields contained in the files the stream references.

The first three pieces of information defining a stream are closely related. For a “one-shot” stream that will be read or written only once, e.g., at model startup, the stream is necessarily only connected to the single file that contains (or will contain, in the case of an output stream) the fields associated with the stream. However, in the case that a stream will be read or written periodically, the user can choose to write have a new file created periodically based on the interval between files. In this case, a single filename is clearly not sufficient, and the user must specify a template to describe the names of all files associated with the stream. It is expected that this filename template will be based on the timestamp of the first record stored in each file, and the stream implementation will therefore define variables that, when used in a stream’s filename, will substitute for the year, month, day, hour, minute, and second, etc., of the first time contained in the file.

Requirements

The proposed stream functionality in MPAS will meet the requirements itemized below. For the 3.0 release of MPAS, the objective is to permit streams to be defined in the Registry file of a core at build time; after the 3.0 release, this capability will be extended to allow streams to be defined or modified at run-time as well.

Definition of Streams

1. Support for streams that are input-only, output-only, both input and output, and neither input nor output.

2. Ability to specify a filename template to describe the files associated with a stream

   When the fields read or written by a stream span multiple files, a single filename is insufficient to identify this set of files; the stream implementation should provide a set of variables that can be used in a filename *template* to describe the set of files associated with the stream.

   At a minimum, the the following variables should be permitted for use in filename templates:

   $Y - Expands to an integer representing a year; can be more than 4 digits
   $M - Expands to a 2 digit month
   $D - Expands to a 2 digit day of the month
   $d - Expands to a 3 digit day of year
   $h - Expands to a 2 digit hour
   $m - Expands to 2 digit minute
   $s - Expands to 2 digit second
   $G - Expands to the grid ID (for multigrid solvers, or for solvers that operate on two more different meshes)

3. Ability to define default streams at build-time

   Developers of an MPAS core should be able to set up a default set of I/O streams in the core's Registry.xml file; these are the streams that will be active unless modified at run-time by a user.

4. Ability to add/remove fields from a stream at run-time

   Ultimately, the user of an MPAS model must be able to modify the set of fields in streams at run-time. For the 3.0 release, we require only the ability to modify streams at build-time by modifying the stream definitions provided in the Registry file of the MPAS model.

5. Ability to define new streams at run-time

   In the current MPAS infrastructure, three streams are available for use by model developers in the Registry: input, output, and restart. For the 3.0 release, we require the ability to define an arbitrary number of streams in the Registry that can be either read or written — once, periodically, or at a set of arbitrary specified times — during model execution. In future, this build-time functionality will be extended to the run-time definition of new streams by model users.

6. Ability to activate streams with packages

   Some models may need to conditionally read or write a set of fields. To support this need, packages may be attached to a stream; a stream is only read or written if any of the packages attached to it is active during the execution of the model.

Timing of Stream Input and Output

7. Ability to attach an arbitrary set of alarms to each direction (input, output) of a stream

   A typical stream may be read or written once (e.g., at model startup), or at a fixed, constant interval throughout the execution of a core. However, future applications of MPAS may require the ability to read or write a stream at a set of aperiodic points in time, possibly in addition to reading or writing at a constant interval. Additionally, a core may need to read a stream at different times from those at which a stream is written. Streams may be read or written when any of the alarms attached to the input or output direction of the stream, respectively, are ringing.

8. Ability to modify the alarms attached to a stream during execution of a core

   Within a core, a developer can have the ability to overwrite the interval associated with the alarm controlling the stream.

9. Ability to force a stream to be handled.

Within a core, a developer can have an arbitrary method of triggering a stream to be read or written. This may be independent of any associated intervals.

10. The ability to read and write from arbitrary times in a stream

    A core can specify a time for the framework to read from, and framework should do its best to find the associated record and either read from, or write to it. Note that, since a stream encompasses all files that match the filename template of the stream, reading from future or past times in a stream may imply the closing of one file and the opening of another file associated with the stream.

11. Handle the reading of times that do not exactly match available times in the files associated with the stream

    The developer of a core should be able to direct the stream to read from a time that is not an exact match for the requested time, if the requested time does not exist in any file associated with the stream. Options for choosing the non-matching time should include “the latest time before the requested time” and “the earliest time after the requested time”.

File Metadata and Standards Compliance

12. Standardized file definitions

    Files for use with an MPAS core are expected to have:

    A time dimension, which is the only unlimited dimension.
    A time variable (xtime) which is monotonically increasing with record number.

13. Support metadata writes as defined from Registry to allow CF compliance.

    Registry should allow the definition of CF compliance meta data attributes. Including: long_name, short_name, units, missing_value.

14. Uniform global attributes for provenance

    All files written by the MPAS framework should contain identical global attributes. Including things like mesh_spec, core, git_version, and all namelist options.

Other Requirements

15. Descriptive error checking of streams

    The MPAS stream implementation should provide as much feedback as possible to developers and users regarding possible errors in the definition of a stream; at a minimum, the implementation should:
    a. Provide a warning (but not a fatal error) if a non-existent field is added to a stream
    b. Halt with a fatal error if a user attempts to modify a restart or initial condition stream at run-time
    c. Halt with a fatal error if two different output streams share a common filename template
    d. Provide a warning if an output stream has no specified output period, and write the stream only once
    e. Halt with a failure if a core has turned on a “no-clobber” option, but the operation would clobber a file or record.

16. Provide a well-defined order for reading input streams

    When reading a collection of input streams, the order in which these streams are read should be deterministic and user-specifiable; for example, this requirement can be met by requiring that streams are read and written in the order of their definition in the Registry or run-time configuration file.

17. Construction of streams handled automatically by infrastructure

    The implementation should provide for the construction of streams, attachment of timers to streams, and other details, based purely on the specification of the streams provided in the Registry file, or in an eventual run-time stream control file, without further code from a developer. Model developers should not have to write code in their cores to assemble streams based on information provided in the Registry.

18. Ability to define the set of fields in a stream using existing Registry constructs

    In order to produce more concise definitions of streams, a developer or user should be able to add entire an entire var_struct, var_array, var, or stream to the set of fields in a stream. In the case that a stream (*stream_A*) is added to another stream (*stream_B*), only the vars, var_structs, and var_arrays of *stream_A* will be added to *stream_B*; no stream included in *stream_A* will be included in *stream_B* to avoid potentially circular dependencies. This requirement allows a stream to be defined using any existing

method for grouping variables together within a Registry file.

19. Ability to specify to/from which time level to read/write a stream for fields with multiple time levels

    The developer of a model must be able to specify, e.g., that a particular read of a stream should place the fields in the second time level of multi-time-level fields. Similarly, it must be possible to cause writes of a stream to write fields from a specified time level. The decision to read/write a field to/from a particular time level depends on the way in which a core is coded; therefore, this decision is not something that should be left up to users, but should be made available to developers making calls to read or write a stream in code.

20. Control modification or loss of existing data on disk

    At runtime, it must be possible to specify whether a stream is allowed to overwrite or modify existing data on disk. This capability should be flexible enough to accommodate various degrees of modification. For example, some instances of a model may be permitted to overwrite or modify any data; some may be permitted to write to existing files only after the files are truncated; some may only be permitted to append to files but not to overwrite existing records; and others may not be permitted to make any sort of addition or modification to existing files.

21. Ability to read / write files containing a non constant number of records.

    The user of a stream must be able to configure the stream manager such that it can read and write files that contain a non constant number of time records.

    An example of this would be daily input or output in monthly files, using a gregorian calendar. Another example is the use of adaptive time steps.

22. Creation of non-existing directory when writing streams.

    The stream manager should identify streams that contain a directory structure that is non-existent and create the relevant directory structure, to prevent issues when writing the stream.

Design

The requirements of the previous section imply two broad needs from the design. Firstly, the design must provide a mechanism for defining streams, both at build-time and, eventually, at run-time. Secondly, the design must provide an interface for core developers to read and write the streams that were defined using this mechanism. In this section one subsection is devoted to each of these two needs. The details of the code implementation of user-configurable I/O streams is the concern of the Implementation section of this document.

A note about timestamps:
Timestamps are listed in this document as “YYYY-MM-DD_hh:mm:ss”, however there are many acceptable formats for a timestamp. While each portion of a timestamp is listed as a number of characters, one can specify an arbitrary number of digits for a timestamp. For example, the year could be specified by 8 digits instead of 4. In addition, portions of a timestamp can be removed if they are not needed. Generally this removal starts from largest and ends with smallest. Below are some examples of acceptable timestamp formats:

“YYYY-MM-DD_hh:mm:ss
“MM-DD_hh:mm:ss”
“DD_hh:mm:ss”
“hh:mm:ss”
“mm:ss”
“ss”
“DD_mm:ss”
“DD_ss”
“YYYY-MM-DD_ss”
“YYYY-MM-DD_mm:ss”

In all of these possible formats, the following definitions are used:
YYYY - year
MM - month
DD - day
hh - hour
mm - minute
ss - second

Stream Definition

Streams will be defined within a Registry.xml file. The following can be used to define a stream based on the requirements from the previous sections.

```
<streams>
        <stream name=”stream1” type=”input”
                  filename_template=”input.$Y-$M-$D.nc”
                  filename_interval=”00-01-00_00:00:00”
                  reference_time=”2000-01-01_00:00:00”
                  interval=”00-00-01_00:00:00” immutable=”true”
                  packages=”packageA;packageB”
                  />
                <var name=”field1”/>
                <var name=”field2”/>
                <var name=”field2”/>
                <var_struct name=”mesh” />
                <var_array name=”array1” />
                <stream name=”stream2” />
        </stream>
</streams>
```

The definitions of the attributes and blocks from the above definition are as follows:

- Attributes:
    - **name (Required)**
      This defines the identifier for the stream. It can be used throughout the Registry.xml file and within the actual model to refer to this specific stream. It must be unique in that no other stream can have the same name.

    - **type (Required)**
      This defines the type of stream. It can take the values “input”, “output”, “none”, and “input;output” (or “output;input”).

    - **filename_interval (Required)**
      This defines the interval between filenames. This tells the stream manager expected spacing between the timestamps used to expand the filename template. It allows the stream manager to handle logic internally to find a previous or next file when searching for a new time, and additionally it allows arbitrary collections of timestamps to be written to files.

      Possible values are “YYYY-MM-DD_hh:mm:ss”, “none”, “input_interval”, or “output_interval”

      See the section “A note about timestamps” above for formatting of the timestamp

option.

A value of “none” causes all records to be written into a single file using reference_time.

A value of “input_interval” or “output_interval” causes the “filename_interval” to take a value equal to the interval listed. i.e. “output_interval” causes “filename_interval” to be equal to “output_interval”.

- **filename_template (Required)**
  This defines the template for files for the stream. It has the following template options:
  - $Y - Expands to a year identifier
  - $M - Expands to a month identifier
  - $D - Expands to a day identifier
  - $d - Expands to a day of year identifier
  - $h - Expands to a hour identifier
  - $m - Expands to a minute identifier
  - $s - Expands to a second identifier
  - $G - Expands to the grid ID (for multigrid solvers, or for solvers that operate on two more different meshes)

  If the filename_template contains a directory structure, and the type of the stream is output, at initialization the stream manager will examine the directory structure.

  If it exists, but contains the incorrect permissions, the model will fail and write an error message.

  If if does not exist, the model will create the directory structure with permissions based on the user’s umask.

- **input_interval (Required for streams containing type “input”)**
  This defines the standard interval for an input stream to be automatically handled. It can take the values of:

  “YYYY-MM-DD_hh:mm:ss”, “none”, “initial_only”

  For timestamp formats see “A note about timestamps”

  none means the stream will not be handled at any automatic interval.

  initial_only means the stream will be automatically handled the first time the stream manager’s read routine is called.

  A value of initial_only cannot be used when filename_interval is set to “input_interval”

- **output_interval (Required for streams containing type “output”)**
  This defines the standard interval for an output stream to be automatically handled. It can take the values of:

  “YYYY-MM-DD_hh:mm:ss”, “none”, “initial_only”

  For timestamp formats see “A note about timestamps”

  none means the stream will not be handled at any automatic interval.

  initial_only means the stream will be automatically handled the first time the stream manager’s write routine is called.

  A value of initial_only cannot be used when filename_interval is set to “output_interval”

- **immutable (Optional)**
  This can be attached to streams so that a stream definition that occurs in a Registry.xml file cannot be modified at run-time.

  Available values are “true” and “false”
  Default value is “false”

- **packages (Optional)**
  Allows packages to be attached to a stream. A stream will be active only if at least one of the packages attached to it is active, or if no packages at all are attached.

Default value is no packages
To attach packages, provide a semi-colon delimited list of package names.

- **precision (optional)**
  Allows a stream to change the precision of the fields written to a file or read from a file.

  Available values are “single” and “double”
  Default value is “double”

- **record_interval (optional)**
  Allows specification for the interval between records in a file.

  Available values are “YYYY-MM-DD_hh:mm:ss” and “none”
  Default value is “none”

  For timestamp formats see “A note about timestamps”

  A value of “none” causes the record interval to be set to the minimum alarm interval for any attached alarm.

- **reference_time (optional)**
  Allows definition of the first time stamp in any file. This determines when filename breaks occur, and is used when searching for input or writing output to ensure the correct file is opened.

  Available values are “YYYY-MM-DD_hh:mm:ss” and “initial_time”
  Default value is “initial_time”

  For timestamp formats see “A note about timestamps”

  A value of “initial_time” causes the reference_time to default to the start time of the simulation clock attached to the stream manager.

- **clobber_mode (optional)**
  Allows specification of how a stream should handle the case where it attempts to write to a file that already exists.

  Available values are “overwrite”, “truncate”, “replace_files”, “append”, or “never_modify”
  Default value is “never_modify”

A value of “overwrite” will allow a stream to overwrite existing records in a file.

A value of “truncate” or “replace_files” will delete existing files, and replace them with newly generated data (resetting the number of records to 0 to begin with).

A value of “append” will allow records to be appended to the end of the file, if it exists.

A value of “never_modify” will not perform any actions to existing files, but new files will be allowed to be created.

- Blocks
    - **streams**
      This is the largest grouping of all streams. Every stream should be defined within one of these blocks

    - **stream**
      This is the definition of a particular stream. Every stream should have one of these blocks.

    - **var** (Child)
      This associates a particular variable with a stream.

    - **var_struct** (Child)
      This associates an entire var_struct with a stream.

    - **var_array** (Child)
      This associates an entire var_array with a stream.

    - **stream** (Child)
      This associates another stream with a stream. It will include all explicitly defined var, var_array, and var_struct blocks that are attached the other stream. Streams nested within the other stream will not be included to avoid circular dependencies.

In addition to adding variables to the definition of a stream, core developers may also attach streams to variables, much as packages may be attached to variables. This alternative method for including a variable in a stream will only work for build-time (i.e., “default”) definition and modification of streams; there is no plan to support attaching streams to variables at run-time.

```
<streams>
      <stream name=”diagnostics” type=”output”
               filename_template=”diagnostics.$Y-$M-$D.nc”
```

```
                filename_interval="01_00:00:00"
                output_interval="1:00:00" immutable="true"
                packages="packageA;packageB"
                />
        </stream>
        <stream name="history" type="output"
                filename_template="history.$Y-$M-$D_$h.$m.$s.nc"
                filename_interval="00_00:00:01"
                output_interval="12:00:00" immutable="true"
                />
        </stream>

</streams>

<var struct name="diag" time_levs="1">
        <var name="uZonal"   type="real"   dimensions="nVertLevels nCells"
              units="m s^{-1}"   description="Zonal wind component"
              streams="diagnostics;history"
        </var>
</var_struct>
```

Run-time Stream Control (for run-time definable streams)

Besides allowing core developers to define default streams in the Registry at build-time, it must not only be possible for users to modify certain aspects of those streams — the filename template for the stream, the number of frames per file, and the read/write interval for the stream — at run-time, but to also modify the fields that belong to a stream and to define entirely new streams. In order to leverage existing use of XML in MPAS, all run-time modification and definition of streams will use the XML schema described in this section.

For each default stream defined in a core's Registry.xml file, a stream tag will be written to an I/O control file that is generated at build time. This stream tag will allow users to modify the filename template of the stream, the filename interval, and the interval at which the stream is read or written. In addition, users can modify the reference time at run time.

```
<?xml version="1.0"?>
<streams>
        <stream name="history"
                type="output"
                filename_template="history.$Y-$M-$D.nc"
                filename_interval="01_00:00:00"
                output_interval="6:00:00"
                />
        </stream>
</streams>
```

Users may define new streams by adding a new stream tag; the name attribute of the stream must be unique, and a type attribute, either ‘input’, ‘output’, ‘input_output’, or ‘none’ must be provided. The list of fields that belong to the new stream is specified as a sequence of one or more of the tags: var, var_struct, stream.

```
<streams>
      <stream name="diagnostics"
              type="output"
              filename_template="diagnostics.$Y-$M-$D.nc"
              filename_interval="01_00:00:00"
              output_interval="1:00:00"
              />
            <var name="uZonal"/>
            <var name="uMeridional"/>
      </stream>
      <stream name="tendencies"
              type="output"
              filename_template="tendencies.$Y-$M-$D.nc"
              filename_interval="01_00:00:00"
              output_interval="1:00:00"
              />
            <var_struct name="tend"/>
      </stream>
      <stream name="hurricane_diagnostics"
              type="output"
              filename_template="tc_diag.$Y-$M-$D.nc"
              filename_interval="01_00:00:00"
              output_interval="1:00:00"
              packages="tc_diagnostics"
              />
            <stream name="diagnostics"/>
            <var name="zoml"/>
            <var name="c_d"/>
            <var name="c_k"/>
      </stream>
</streams>
```

In the example above, the “hurricane_diagnostics” stream is only written if the “tc_diagnostics” package in the model is active; this stream includes all fields listed explicitly as belonging to the “diagnostics” stream, plus the fields “zoml”, “c_d”, and “c_k”.

Attributes of immutable streams are also run-time controllable. This is because at run-time a user might want to modify the interval a stream is read / written or what the filename template is for a particular stream.

Immutable streams are controlled similarly to other streams, i.e.:

```
<streams>
      <immutable_stream name=”input” type=”input”
            filename_template=”core_input.nc”
            input_interval=”initial_only”/>
      <immutable_stream name=”restart”
            type=”input;output”
            filename_template=”core_restart.$Y-$M-$D.nc”
            output_interval=”0000-00-01_00:00:00”
            input_interval=”initial_only”
            />
</streams>
```

Immutable streams are not allowed to modify their contents at run time (variables, variable structures, and streams that are inside the stream).

While controlling a stream at run-time, users have available the following XML attributes and blocks to modify.

- Attributes:
    - **name (Required)**
      This defines the identifier for the stream. It can be used throughout the Registry.xml file and within the actual model to refer to this specific stream. It must be unique in that no other stream can have the same name.

    - **type (Required)**
      This defines the type of stream. It can take the values “input”, “output”, “none”, and “input;output” (or “output;input”).

    - **filename_interval (Required)**
      This defines the interval between filenames. This tells the stream manager expected spacing between the timestamps used to expand the filename template. It allows the stream manager to handle logic internally to find a previous or next file when searching for a new time, and additionally it allows arbitrary collections of timestamps to be written to files.

      Possible values are “YYYY-MM-DD_hh:mm:ss”, “none”, “input_interval”, or “output_interval”

      See the section “A note about timestamps” above for formatting of the timestamp option.

      A value of “none” causes all records to be written into a single file using reference_time.

A value of “input_interval” or “output_interval” causes the “filename_interval” to take a value equal to the interval listed. i.e. “output_interval” causes “filename_interval” to be equal to “output_interval”.

- **filename_template (Required)**
  This defines the template for files for the stream. It has the following template options:
  - $Y - Expands to a year identifier
  - $M - Expands to a month identifier
  - $D - Expands to a day identifier
  - $d - Expands to a day of year identifier
  - $h - Expands to a hour identifier
  - $m - Expands to a minute identifier
  - $s - Expands to a second identifier
  - $G - Expands to the grid ID (for multigrid solvers, or for solvers that operate on two more different meshes)

  If the filename_template contains a directory structure, and the type of the stream is output, at initialization the stream manager will examine the directory structure.

  If it exists, but contains the incorrect permissions, the model will fail and write an error message.

  If if does not exist, the model will create the directory structure with permissions based on the user’s umask.

- **input_interval (Required for streams containing type “input”)**
  This defines the standard interval for an input stream to be automatically handled. It can take the values of:

  “YYYY-MM-DD_hh:mm:ss”, “none”, “initial_only”

  For timestamp formats see “A note about timestamps”

  none means the stream will not be handled at any automatic interval.

  initial_only means the stream will be automatically handled the first time the stream manager’s read routine is called.

  A value of initial_only cannot be used when filename_interval is set to “input_interval”

- **output_interval (Required for streams containing type “output”)**
  This defines the standard interval for an output stream to be automatically handled. It can take the values of:

  “YYYY-MM-DD_hh:mm:ss”, “none”, “initial_only”

  For timestamp formats see “A note about timestamps”

  none means the stream will not be handled at any automatic interval.

  initial_only means the stream will be automatically handled the first time the stream manager’s write routine is called.

  A value of initial_only cannot be used when filename_interval is set to “output_interval”

- **packages (Optional)**
  Allows packages to be attached to a stream. A stream will be active only if at least one of the packages attached to it is active, or if no packages at all are attached.

  Default value is no packages
  To attach packages, provide a semi-colon delimited list of package names.

- **precision (optional)**
  Allows a stream to change the precision of the fields written to a file or read from a file.

  Available values are “single” and “double”
  Default value is “double”

- **record_interval (optional)**
  Allows specification for the interval between records in a file.

  Available values are “YYYY-MM-DD_hh:mm:ss” and “none”
  Default value is “none”

  For timestamp formats see “A note about timestamps”

  A value of “none” causes the record interval to be set to the minimum alarm interval for any attached alarm.

- **reference_time (optional)**
  Allows definition of the first time stamp in any file. This determines when filename breaks occur, and is used when searching for input or writing output to ensure the correct file is opened.

  Available values are “YYYY-MM-DD_hh:mm:ss” and “initial_time”
  Default value is “initial_time”

  For timestamp formats see “A note about timestamps”

  A value of “initial_time” causes the reference_time to default to the start time of the simulation clock attached to the stream manager.

- **clobber_mode (optional)**
  Allows specification of how a stream should handle the case where it attempts to write to a file that already exists.

  Available values are “overwrite”, “truncate”, “replace_files”, “append”, or “never_modify”
  Default value is “never_modify”

  A value of “overwrite” will allow a stream to overwrite existing records in a file.

  A value of “truncate” or “replace_files” will delete existing files, and replace them with newly generated data (resetting the number of records to 0 to begin with).

  A value of “append” will allow records to be appended to the end of the file, if it exists.

  A value of “never_modify” will not perform any actions to existing files, but new files will be allowed to be created.

- Blocks
  - **streams**
    This is the largest grouping of all streams. Every stream should be defined within one of these blocks. It should be generated by default in the streams configuration file generated during build.

  - **immutable_stream**
    This is the definition of a particular immutable stream. Every stream defined this way should appear in the core’s Registry.xml file, and it’s contents are not configurable at run-time. Adding child blocks within an immutable_stream causes

run-time errors and should be avoided.

- **stream**
  This is the definition of a particular stream. Every stream should have one of these blocks.

- **var** (Child)
  This associates a particular variable with a stream.

- **var_struct** (Child)
  This associates an entire var_struct with a stream.

- **var_array** (Child)
  This associates an entire var_array with a stream.

- **stream** (Child)
  This associates another stream with a stream. It will include all explicitly defined var, var_array, and var_struct blocks that are attached the other stream. Streams nested within the other stream will not be included to avoid circular dependencies.

Stream Handling

The handling of streams defined in a core's Registry.xml will be accomplished through a single module that provides interface routines for reading streams, writing streams, modifying the alarms for streams, and forcing streams to be read or written. Additionally, this module will provide an interface for the construction of the streams contained in the module, to be used by the registry or other infrastructure code for building streams based on compile-time or runtime definitions.

In order to avoid future namespace conflicts between different MPAS cores, the set of streams defined by a core will be associated with a stream manager derived type. The stream manager module will be otherwise stateless, and will contain no module variables.

The following constants may be used to indicate the 'direction' of new streams defined in the stream manager module:

- `MPAS_STREAM_INPUT` : used to indicate a stream that can be read
- `MPAS_STREAM_OUTPUT` : used to indicate a stream that can be written
- `MPAS_STREAM_INPUT_OUTPUT` : used to indicate a stream that can be both read and written
- `MPAS_STREAM_NONE` : used to indicate a stream that can be neither read nor written

Streams defined in the stream manager module may have ‘properties’ associated with them; the following constants will be defined to allow for streams that cannot be further modified at run-time, or that are not part of an active package:

- MPAS_STREAM_IMMUTABLE : used to indicate that the set of fields in the stream cannot be modified
- MPAS_STREAM_MUTABLE : used to indicate that fields can be added or removed from the stream
- MPAS_STREAM_ACTIVE : used to indicate an active stream, which will be read or written when a stream manager call is made to read or write streams
- MPAS_STREAM_INACTIVE : used to indicate an inactive stream, which will be neither read nor written when a stream manager call is made to read or write streams

For the purposes of determining which time record to read from streams that contain multiple input records, and for which the user has requested to read at a time that is not an exact match in the stream, the following constants will be provided:

- MPAS_STREAM_NEAREST : the time nearest to the requested time will be read; in case two times are equally near to the requested time, the earlier of the two times will be read
- MPAS_STREAM_EXACT_TIME : fields will only be read if an exact match is found; otherwise, no reading will take place and an indication will be somehow provided to the user
- MPAS_STREAM_LATEST_BEFORE : the latest time in the stream that is not later than (but may be equal to) the requested time will be read
- MPAS_STREAM_LATEST_STRICTLY_BEFORE : the latest time in the stream that earlier than but not equal to the requested time will be read
- MPAS_STREAM_EARLIEST_AFTER : the earliest time in the stream that is not earlier than (but may be equal to) the requested time will be read
- MPAS_STREAM_EARLIEST_STRICTLY_AFTER : the earliest time in the stream that is later than but not equal to the requested time will be read

The stream manager module will provide the following public interface routines.

```
!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_init
!
!> \brief Initialize a new MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Instantiates and initializes a streamManager type with a timekeeping
!>  clock and a pool from which fields may be drawn and added to streams.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_init(manager, clock, allFields, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      type (MPAS_Clock_type), intent(inout) :: clock
      type (MPAS_Pool_type), intent(in) :: allFields
```

```
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_finalize
!
!> \brief Free all memory associated with an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Destroys a streamManager type, freeing all memory that was created as
!>  part of the manager; the external clock and field pool associated with
!>  the streamManager are unaffected.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_finalize(manager, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_create_stream
!
!> \brief Instantiate a new stream within an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Creates a new stream within the stream manager. The 'direction'
!>  argument may be either MPAS_STREAM_INPUT, MPAS_STREAM_OUTPUT,
!>  MPAS_STREAM_INPUT_OUTPUT, or MPAS_STREAM_NONE. The 'filename' argument
!>  is the template of the filenames that are associated with the stream.
!>  The 'filenameInterval' argument tells the stream manager how often a new
!>  file should be created. Knowing how often files occur, and
!>  the filename template, a 'referenceTime' argument must be provided to
!>  specify the first timestamp appearing in any of the files associated with
!>  the stream, thereby determining where the "file breaks" will occur between
!>  timestamps. If no 'referenceTime' is specified, the start time of the
!>  clock associated with the stream handler will be used as the reference
!>  time.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_create_stream(manager, streamID, direction,
filename, filenameInterval, referenceTime, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      integer, intent(in) :: direction
```

```
      character (len=StrKIND), intent(in) :: filename
      character (len=StrKIND), intent(in) :: filenameInterval
      type (MPAS_Time_type), intent(in), optional :: referenceTime
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_destroy_stream
!
!> \brief Free all memory associated with a stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_destroy_stream(manager, streamID, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_set_property
!
!> \brief Sets a property of a stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_set_property(manager, streamID, property, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      integer, intent(in) :: property
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_get_property
!
!> \brief Queries the status of a stream property in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
```

```
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_get_property(manager, streamID, property, isSet,
ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      integer, intent(in) :: property
      logical, intent(out) :: isSet
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_add_pkg
!
!> \brief Attach a package logical to the specified stream.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_add_pkg(manager, streamID, package, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      logical, intent(in), target :: package
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_remove_pkg
!
!> \brief Detaches a package logical from the specified stream.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_remove_field(manager, streamID, package, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      logical, intent(in), target :: package
      integer, intent(out), optional :: ierr
```

```
!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_add_field
!
!> \brief Add a field to the specified stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_add_field(manager, streamID, fieldName, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      character (len=StrKIND), intent(in) :: fieldName
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_remove_field
!
!> \brief Remove a field from the specified stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_remove_field(manager, streamID, fieldName, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      character (len=StrKIND), intent(in) :: fieldName
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_add_att
!
!> \brief Add an attribute to the specified stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_add_att(manager, streamID, att, fieldName, ierr)
```

```
      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      real (kind=RKIND), intent(in) :: att
      character (len=StrKIND), intent(in) :: fieldName
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_add_alarm
!
!> \brief Add an I/O alarm to a stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_add_alarm(manager, streamID, alarm, alarmID,
direction, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      type (MPAS_Alarm_type), target, intent(in) :: alarm
      integer, intent(in) :: alarmID
      integer, intent(in) :: direction
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_remove_alarm
!
!> \brief Remove an I/O alarm from a stream in an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  Blah.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_remove_alarm(manager, streamID, alarmID, direction,
ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in) :: streamID
      integer, intent(in) :: alarmID
      integer, intent(in) :: direction
      integer, intent(out), optional :: ierr
```

```
!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_write
!
!> \brief Write streams that are managed by an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  With no optional arguments, writes all streams whose alarms are ringing.
!>  The 'streamID' argument optionally specifies the ID of a particular stream
!>  to be written; if no other optional arguments are given, the specified
!>  stream is only written if any of its alarms are ringing.
!>  The 'timeLevel' argument optionally specifies, for fields with multiple
!>  time levels, the time level from which fields should be written.
!>  The 'mgLevel' argument optionally specifies, for fields that exist for
!>  multiple grid levels, the grid level from which fields should be written.
!>  The 'forceWriteNow' argument optionally specifies that all streams -- or
!>  the stream specified by the 'streamID' argument -- should be written by
!>  the call regardless of whether any alarms associated with the stream(s)
!>  are ringing.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_write(manager, streamID, timeLevel, mgLevel,
forceWriteNow, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in), optional :: streamID
      integer, intent(in), optional :: timeLevel
      integer, intent(in), optional :: mgLevel
      logical, intent(in), optional :: forceWriteNow
      integer, intent(out), optional :: ierr


!-----------------------------------------------------------------------
!  routine MPAS_stream_mgr_read
!
!> \brief Read streams that are managed by an MPAS stream manager.
!> \author Michael Duda, Doug Jacobsen
!> \date   13 June 2014
!> \details
!>  With no optional arguments, reads all streams whose alarms are ringing.
!>  The 'streamID' argument optionally specifies the ID of a particular stream
!>  to be read; if no other optional arguments are given, the specified stream
!>  is only read if any of its alarms are ringing.
!>  The 'timeLevel' argument optionally specifies, for fields with multiple
!>  time levels, the time level into which fields should be read.
!>  The 'mgLevel' argument optionally specifies, for fields that exist for
!>  multiple grid levels, the grid level into which fields should be read.
```

```
!>  The 'when' argument optionally specifies the timestamp from which fields
!>  are to be read.
!>  The 'whence' argument optionally specifies the method for determining
!>  the timestamp to read from in case an exact match is not found for the
!>  read timestamp, which is the current time unless the optional 'when'
!>  argument is given; possible values are MPAS_STREAM_EXACT_TIME,
!>  MPAS_STREAM_NEAREST, MPAS_STREAM_LATEST_BEFORE,
!>  MPAS_STREAM_LATEST_STRICTLY_BEFORE, MPAS_STREAM_EARLIEST_AFTER, or
!>  MPAS_STREAM_EARLIEST_STRICTLY_AFTER.
!>  The optional output argument 'actualWhen' returns the actual time read
!>  from a stream in case an exact match for the 'when' time is not found,
!>  and a nearby time is selected using the 'whence' argument.
!
!-----------------------------------------------------------------------
subroutine MPAS_stream_mgr_read(manager, streamID, timeLevel, mgLevel,
rightNow, when, whence, actualWhen, ierr)

      type (MPAS_streamManager_type), intent(inout) :: manager
      character (len=StrKIND), intent(in), optional :: streamID
      integer, intent(in), optional :: timeLevel
      integer, intent(in), optional :: mgLevel
      logical, intent(in), optional :: rightNow
      character (len=StrKIND), intent(in), optional :: when
      integer, intent(in), optional :: whence
      character (len=StrKIND), intent(out), optional :: actualWhen
      integer, intent(out), optional :: ierr
```

File Specification

Files to be read in by the stream manager must conform to the following set of requirements:

- Have a dimension named Time which is equal to the number of records in the file (and is unlimited)
- Have a dimension named StrLen which is equal to 64 and provides the maximum length of each timestamp in the xtime variable
- Have a variable xtime which is the time stamp for each time record, and must be monotonically increasing with Time record. It's dimensions should be (Time, StrLen)

Example Usage

Here we provide some examples of how the stream manager interface described above may be employed.

Implementation

To be determined.