MeasurementSet access in AIPS++

Athol Kemball

October 15, 2000

Contents

1 Introduction

The C++ MeasurementSet (MS) access classes are fundamental to the AIPS++ library, and find general use in several areas, including data fillers, the `ms` tool and a range of data reduction applications. This note is focused primarily on basic MS access in synthesis at this time; it describes the design of the current classes, and considers their future extension and consolidation. The MS data format is described in AIPS++ Note 229.

2 Requirements for the MS access classes

MeasurementSets are AIPS++ `Tables` and their existing access classes are built closely on the `Table` infrastructure. This assumption is implicit in the requirements listed in this section. The C++ classes used for MS access need to meet the following requirements, or provide the following services:

1. **Basic whole MS operations:** MS creation, deletion and related MS-level operations.
2. **Column and row access:** Access to MS MAIN and sub-table rows and columns.
3. **Sorting and selection:** The ability to support bulk sorting and selection operations. MS selections are expected to be expressed in a close variant of the Table Query Language (TaQL), defined elsewhere.

4. **MAIN MS iteration:** Sequential iteration through the MAIN MS table on arbitrary iteration indices, with optional retrieval of associated MS sub-table data matching the current MAIN table iteration interval. Indexed access to the MAIN table, on limited key indices, is a secondary requirement. MAIN table iteration includes simultaneous or sequential iteration through multiple MS considered as a set. Iteration is also required within MS MAIN rows, in frequency, velocity or polarization.

5. **MS sub-table iteration:** Independent sequential iteration through individual MS sub-tables, on arbitrary iteration indices. Keyed access to MS sub-tables in read-write mode, on arbitrary key indices. Iteration is also required within rows, in frequency, velocity or polarization, as appropriate for individual sub-tables.

6. **MS derived quantities:** Computation or retrieval of derived MS quantities, which require data access across the MS as a whole. Examples include elevation computation, assembling higher-level coordinate information or retrieval of Doppler shift parameters.

7. **MS data ranges:** Determination of the data ranges in a given MS, for any MAIN or sub-table columns or derived quantities.

8. **MS data objects:** Provision of C++ classes to model a data cube containing part or all of the current iteration or selection, which can be operated upon by other agent classes (e.g. calibration), and passed between AIPS++ classes and between C++ and Glish.

9. **Connection to Glish:** Versatile access to the C++ MS access classes from Glish.

10. **Scratch column management:** Creation and deletion of scratch columns which may be added to, or removed from the MS during reduction.

11. **Parallel I/O support:** Full integration of parallel I/O capabilities within the standard serial MS access classes, and the ability to optionally enable the parallel I/O capabilities.

12. `Table` **infrastructure use:** Full re-use of all `Table` system infrastructure and design philosophy wherever possible in the MS access classes.

13. **Efficiency:** I/O is a critical part of astronomical data reduction performance, and efficiency is a key requirement for MS access. This includes integrated I/O profiling, in both the serial and parallel case.

3 Current capabilities and design

The following capabilities are currently provided:

1. **Basic whole MS operations:** The basic classes for creating a MS or MS sub-table are: `MeasurementSet`, `MSAntenna`, `MSDataDescription`, `MSDoppler`, `MSFeed`, `MSField`, `MSFlagCmd`, `MSFreqOffset`, `MSHistory`, `MSObservation`, `MSPointing`, `MSPolarization`, `MSProcessor`, `MSSource`, `MSSpectralWindow`, `MSState`, `MSSysCal`, and `MSWeather`. The `MSTable` class is a templated base class for the MS MAIN and sub-tables.

2. **Column and row access:** The table column accessors are provided by `MS*Columns`, and the table column definitions are defined in `MS*Enums`. `MSColumns` provides access to the MS as a whole. *Missing: no specialized MS row-based access.*

3. **Sorting and selection:** MS sorting and selection is possible using the TaQL capabilities directly. Sorting and selection are also performed by `MSSelector`, to some degree in `MSIter`, and in specialized form by reduction classes such as `imager` and `calibrater`, amongst others. *Missing: i) unified MS selection to TaQL converter (partially implemented in* `[calibrater|imager].g`*) allowing aliases, sub-table lookup, derived quantities and [0,1] indexing; ii) centralization of MS selection services in a utility class; and, iii) unification of* `MSSelector` *GlishRecord selection syntax and a general MS selection syntax, including unification of* `MSSelector` *keywords and* `MSCalEnums`.

4. **MAIN MS iteration:** Sequential iteration on limited indices is provided by `MSIter` and `VisibilityIterator`, using `TableIterator`. This includes velocity or frequency iteration within a row, and iteration over rows with the same time stamp within an iteration interval. Sequential iteration through multiple MeasurementSets is supported. Higher-level access to `MSIter` is provided by `MSSelector`. *Missing: i) arbitrary iteration indices; ii) more control over velocity iteration including multi-source support; iii) access to general MS selection; iv) simultaneous, multi-MS iteration; v) retrieval of MS sub-table rows associated with the current MAIN iteration block; vi) write access to more columns; vii) customized sub-iteration within a MAIN iteration block; and viii) indexed MAIN access.*

5. **MS sub-table iteration:** Read-only, indexed access is possible in certain forms using the `MS*Index` classes, which are built on top of `ColumnsIndex`. *Missing: i) arbitrary, read-write keyed lookup, with different forms of interpolation where required; and, ii) specialized sequential iterators.*

6. **MS derived quantities:** Currently provided by several classes, including `MSDerivedValues`, and `MSDopplerUtil`, amongst others. *Missing: other computed quantities as needed.*

7. **MS data ranges:** Currently performed by `MSRange`. *Missing: C++ interface which does not use GlishRecords.*

8. **MS data objects:** Currently provided by `VisBuffer`, but specialized forms also implemented in `MSFlagger`. *Missing: i) greater MS coverage; ii) optional write-through to underlying MS; iii) customized frequency averaging; and, iv) re-select on buffer using standard MS selection syntax.*

9. **Connection to Glish:** Currently provided by the `ms` tool, using `MSSelector, MSSelUtil[2], MSRange, and MSFlagger` primarily. *Missing: i) greater unification with* `VisibilityIterator` *and* `VisBuffer`*; ii) duplicates some MS utility functions implemented elsewhere (e.g. time, frequency averaging).*

10. **Scratch column management:** `VisSet` currently handles creation and addition of the MAIN columns: MODEL_DATA, CORRECTED_DATA and IMAGING_WEIGHT.

11. **Parallel I/O support:** Prototype parallel I/O capabilities exist in `pimager::tryparread`. *Missing: full parallel I/O implementation.*

12. `Table` **infrastructure use:** It is believed that no infrastructure is implemented in the current MS access classes which should properly be moved to the `Table` system. *Missing: i) large file support in the* `Table` *system; and, ii) ability to remove existing columns held in TSM*

13. **Efficiency:** I/O profiling curently is implemented using PABLO (UIUC). *Missing: continued profiling of computational and I/O components.*

4 Proposed revisions

This section considers short-term revisions to the existing MS access classes to rationalize certain existing capabilities, and to add missing capabilities required for application development in several areas in the short term.

1. **Consolidate Glish access layer:** At present, methods to pack output MS data into GlishRecords, and to unpack and accept input data and selections in GlishRecord format, exist in `MSFlagger,` `MSRange` and `MSSelector`. These classes do not use `VisibilityIterator` or `VisBuffer`, but use buffers of GlishRecords to hold the data where required in C++. The proposal in this area is to isolate the GlishRecord interface code in a separate class called `MSGlishData` (or similar), and to migrate to the use of `VisibilityIterator` and `VisBuffer` in the classes mentioned above. This will allow consolidation of code existing elsewhere for data averaging in time and frequency. The Glish interface class will also be responsible for translating between a `VisBuffer` and a Glish data object, which will remain in GlishRecord representation.

2. **Selection:** The proposal in this area is to concentrate MS selection and sorting only in `MSSelector`, including: a) add MS selection to TaQL converter (currently in Glish); b) form union MSCalEnums and MSSelectionKeywords; c) add sorting (col. names); d) use new MSSelector in MSIter, imager and calibrater.

3. **MSIter, VisibilityIterator, and VisBuffer:** The major changes to these classes include addition of access to selection, and sub-table look-up. Specifically: a) add general selection to constructor, which uses new MSSelector; b) add sub-table lookup for current MAIN iterator interval, and propagate these changes to VisIter and VisBuffer.