Simulation-Based Engineering Lab
University of Wisconsin-Madison
Technical Report TR–2017–09

Overview of the Chrono ADAMS Parser

Conlain Kelly and Radu Serban

Dept. of Mechanical Engineering, University of Wisconsin – Madison

January 18, 2018

1 Introduction

This report presents a brief overview of the Chrono [1] functionality for parsing MSC ADAMS [2] `adm` input files. It also includes a basic usage guide with code examples. The parser currently recognizes the following ADAMS structures:

- `PART` (a body and corresponding properties)
- `MARKER` (reference frame that JOINTS attach to)
- `JOINT` (a kinematic constraint between two MARKERS)
- `ACCGRAV` (the direction and magnitude of gravity)
- `GRAPHICS` (visualization assets attached to a marker)

The following are matched and then ignored:

- `REQUEST` (controls what variables are output)
- `UNITS` (sets simulation units)
- `OUTPUT` (controls output file generation)

There are many other ADAMS objects that the parser completely ignores, but those that are interpreted and converted comprise a large subset of the ADAMS functionality.

2 Background

MSC Adams (Automatic Dynamic Analysis of Mechanical Systems) [2] is a proprietary multibody dynamics software used to design and model mechanical systems.

For 3-D multibody system modeling, ADAMS employs a Cartesian (i.e., maximal coordinate) formulation, similar to the one used in Chrono. The main difference between the two is the representation of 3-D rotations: Chrono relies on unit quaternions (four parameters with a normalization constraint), ADAMS uses Euler angles (three independent parameters). Both packages offer exhaustive libraries of kinematic joints and force elements.

Modeling and simulation with ADAMS is usually done through its integrated ADAMS/View GUI. Nonetheless, the interface between the ADAMS modeling module and its simulation module (ADAMS/Solver) is intermediated through so-called `adm` files which are ASCII text files with a proprietary format. Commands for controlling the simulation (such as integrator tolerances and maximum step-size, final simulation time and output frequency, type of output, etc.) are provides separately, into a so-called `acf` file. Examples of `adm` and `acf` files are provided in Appendix A.

ADAMS provides a simple grammar for specifying mathematical functions and referring to states of the underlying system. Such expressions are entered directly in the ADAMS dataset (the `adm` file) using a FORTRAN-like syntax. Note that, for uniformity, ADAMS/View generates function expression even for the simplest modeling elements. For example, the following snippet from an `adm` file represents a linear spring-damper force element:

```
!                          adams_view_name='SPRING_1.sforce'
SFORCE/1
, TRANSLATIONAL
, I = 204
, J = 205
, FUNCTION = - 100.0*(dm(204,205)-2.0)
, - 1.0*vr(204,205)
, + 0.0
!
!                          adams_view_name='SPRING_1.deformation'
VARIABLE/1
, FUNCTION = DM(204,205) - 2.0
!
!                    adams_view_name='SPRING_1.deformation_velocity'
VARIABLE/2
, FUNCTION = VR(204,205)
!
!                          adams_view_name='SPRING_1.force'
VARIABLE/3
, FUNCTION = (DX(204,205)*FX(204,205) +
, DY(204,205)*FY(204,205) +
, DZ(204,205)*FZ(204,205))/
, DM(204,205)
```

Parsing an `adm` file thus requires a lexical analyzer generator.

3 Implementation

The parser makes three passes to parse an `adm` file. A first pass is done over the `adm` file to parse it into a list of tokens. A second pass runs over the set of tokens and compiles them into a set of c++ data structures for each. The final pass then converts these c++ representations into Chrono objects. This 3-pass structure is necessary since ADM files don't necessarily specify objects in a useful order (markers can be declared before their corresponding parts).

The first pass uses `FLEX` [3] to generate a tokenizer to parse the `adm` file. A .lex file specifies regex patterns and a corresponding line of C++ to run if a match occurs. For example, when the tokenizer encounters the token `MARKER`, it will flag that token as a match for a `ChMarker` and add a <token, arguments> pair to a list of such tokens.

The second pass runs over the token stream and determines what kind of Chrono object is needed. The stream consists sets of primary tokens (tokens corresponding to some object like a `PART`) and corresponding attribute tokens. For example, the parser will see that a `MARKER` token was detected and will read the following tokens to extract relevant properties, such as position, orientation, and attached part. In this fashion, it creates lists of parts, markers, joints, and visualization assets, represented as the C++ structures in Listing 1.

The third pass runs through these lists and constructs, in order, the corresponding `ChBodys`, `ChMarkers`, `ChLinks`, and `ChAssets`. This ordering is necessary to ensure that the proper `ChBodys` and `ChMarkers` exist before the joints that reference them are created. This way, each `Chrono` object can be created, initialized, and added to the containing system in one pass.

Listing 1: Intermediate objects

```
struct adams_part_struct {
    bool fixed;                  // Fixed to ground
    double mass;                 // Part mass
    std::string cm_marker_id;    // COM marker
    double loc[3];               // Location of part in global frame
    double rot[3];               // Orientation of part in global frame
    double inertia[6];           // Moments of inertia
};
struct adams_joint_struct {
    std::string type;        // REVOLUTE, TRANSLATIONAL, etc.
    std::string marker_I;    // First constrained marker
    std::string marker_J;    // Second constrained marker
};
struct adams_marker_struct {
    std::string part_id;   // Attached part
    double loc[3];         // Location relative to part
    double rot[3];         // Orientation relative to part
};
```

3.1 Report

The `ChParserADAMS::Report` class provides an interface for the user to access bodies, joints, and forces parsed from the `.adm` file. A report object is created during parsing to store, in maps hashed by the element name, the lists of `Chrono` bodies, joints, and loads. The relevant data structures are shown in Listing 4. The `ChParserADAMS::Report` provides methods for printing the report and for accessing bodies, joints, and loads by their name. Additionally, it provides an interface for users to modify bodies and joints loaded into the system, primarily to add visualization and collision assets. The report for a parser can be accessed via `ChParserADAMS::GetReport()`.

3.2 Visualization

The parser currently reads in 3 visualization assets from the `adm` file: `BOX`, `CYLINDER`, and `ELLIPSOID`. These are only loaded into the system if the parser's `m_visType` flag is set to `true`. The given `visType` structure provides the ability to add a new visualization technique if so desired. Additionally visualization assets can be added to parsed bodies by accessing those bodies via the `Report` class.

Listing 2: ChParserADAMS::Report class

```
/// Report containing information about objects parsed from file
class ChApi Report {
public:
/// Information about a joint read in from ADAMS.
struct JointInfo {
    std::string type;                 ///< joint type as shown in adm file
    std::shared_ptr<ChLink> joint; ///< Chrono link (joint)
};
/// list of body information
std::unordered_map<std::string, std::shared_ptr<ChBodyAuxRef>> bodies;
/// list of joint information
std::unordered_map<std::string, JointInfo> joints;
...
```

Listing 3: ChParserAdams usage example

```
std::string filename = "adams/test_Revolute_Case01.adm";
// Make a system
ChSystemSMC my_system;

// Create parser instance and set options.
ChParserAdams parser;
parser.SetVisualizationType(ChParserAdams::VisType::LOADED);
parser.SetVerbose(true);
parser.Parse(my_system, filename);
```

3.3 Collision and Contact

Currently all bodies are set as non-collision objects. However, bodies can be accessed via the `Report` class (see §3.1), providing the user with an interface to add collision models to bodies after the parser has run.

3.4 Sample Usage

An example of parsing an `adm` file to populate an existing Chrono system is as shown in Listing 3.

The usage pattern is:

1. create parser object;
2. set parsing options;
3. invoke one of the `Parse` methods.

For a complete example of using the ADAMS to Chrono parser, see the `demo_IRR_Adams_parser`

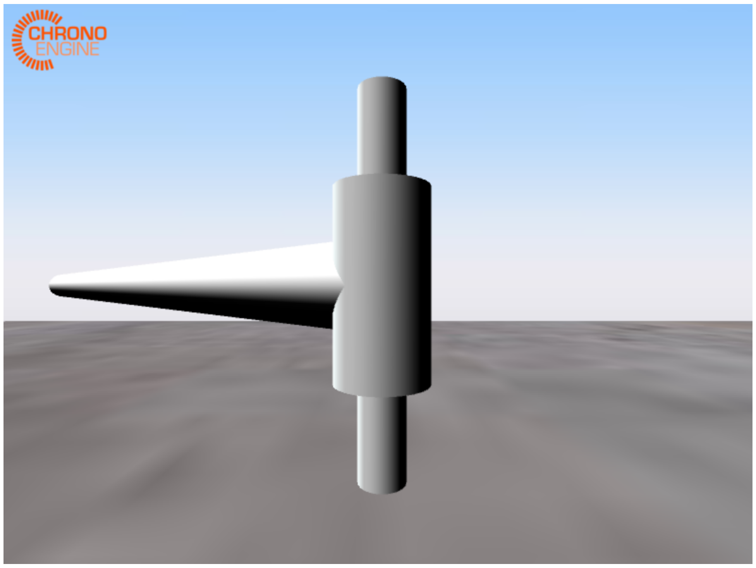

Figure 1: Snapshot from a Chrono simulation of a parsed ADAMS model. The visualization mode was set to LOADED.

Listing 4: ChParserAdams::Report Output from demo_IRR_Adams_parser

```
Parsed 2 bodies:
name: "01"
name: "02"
Parsed 1 joints:
name: "1", type: "REVOLUTE"
```

demonstration program available with the Chrono distribution [4]. A snapshot from the simulation of this translated model is shown in Fig. 1. The output from ChParserADAMS::PrintReport() is provided in Listing 4.

3.5 Current Limitations

- No collision models are read or created at parse-time.
- Unrecognized joints (ADAMS joints outside those implemented in the parser) are flagged with a warning to stderr but then ignored.
- ADAMS generalized constraints (GCONs) are ignored and must be user-implemented.

A ADAMS acf and adm file samples

Listing 5: ADAMS acf file example

```
test_Revolute_Case01.adm
test_Revolute_Case01_ADAMS
output/noseparator
integrator/gstiff, &
error = 1.0e-4, hmax=1e-5
simulate/transient, &
end=5.0, dtout=1.0E-002
stop
```

Listing 6: ADAMS adm file for a simple pendulum system

```
ADAMS/View model name: test_Revolute_Case01
!
!------------------------------ SYSTEM UNITS ------------------------------
!
UNITS/
, FORCE = NEWTON
, MASS = KILOGRAM
, LENGTH = METER
, TIME = SECOND
!
!--------------------------------- PARTS ----------------------------------
!
!--------------------------------- Ground ---------------------------------
PART/01, GROUND
!                       World Coordinate System Marker
MARKER/0101, PART = 01
!              Revolute Joint Attachment Marker
!              (-90 deg rotation about the X axis)
MARKER/0102, PART = 01
, QP = 0, 0, 0
, REULER = 180D, 90D, 180D
!
!
!                            Joint Geometry
MARKER/0103, PART = 01
, QP = 0, -.4, 0
, REULER = 180D, 90D, 180D
!
GRAPHICS/0101
, CYLINDER
, CM = 0103
, LENGTH = .8
, RADIUS = 0.05
!
!
!--------------------------------- Pedulum --------------------------------
!
PART/02, MASS = 1
, CM = 0201, IP = 0.04, 0.1, 0.1
!
!              Pedulum Center Marker
!              (-90 deg rotation about the X axis)
MARKER/0201, PART = 02
, QP = 2, 0, 0
, REULER = 180D, 90D, 180D
!
!              Pedulum Revolute Joint Attachment Marker
!              (-90 deg rotation about the X axis)
MARKER/0202, PART = 02
, QP = 0, 0, 0
, REULER = 180D, 90D, 180D
!
!                            Draw Geometry
!                            Main Pendulum Body
!              (Point Z axis along original x axis)
MARKER/0203, PART = 02
, QP = 0, 0, 0
, REULER = 90D, 90D, 0
!
GRAPHICS/0201
, CYLINDER
, CM = 0203
, LENGTH = 4
, RADIUS = 0.1
!
!                            Joint Cylinder
MARKER/0204, PART = 02
```

```
, QP = 0, -.2, 0
, REULER = 180D, 90D, 180D
!
GRAPHICS/0202
, CYLINDER
, CM = 0204
, LENGTH = .4
, RADIUS = 0.1
!
!------------------------------ CONSTRAINTS ------------------------------
!
!                              Pendulum Revolute Joint
JOINT/1, REVOLUTE
, I = 0102, J = 0202
!
!---------------------------- DATA STRUCTURES ----------------------------
!
!
!------------------------- GRAVITATIONAL ACCELERATION -------------------------
!
ACCGRAV/
, KGRAV = -9.80665
!
!---------------------------- OUTPUT REQUESTS ----------------------------
!
REQUEST/01, D,I=0201,J=0101,C=DISPLACEMENT: X Y Z PSI THETA PHI (body-fixed-3-1-3)
REQUEST/02, V,I=0201,J=0101,C=VELOCITY X Y Z WX WY WZ
REQUEST/03, A,I=0201,J=0101,C=ACCELERATION X Y Z WDX WDY WDZ
REQUEST/04, F2=ORIENT(27,1,0201,0101)\F3=ORIENT(27,2,0201,0101)\F4=ORIENT(27,3,0201,0101)\F6=ORIENT(27,4,0201,0101),C=EULER
    ↪ PARAMETERS
REQUEST/05, F2=JOINT(1,0,2,0)\F3=JOINT(1,0,3,0)\F4=JOINT(1,0,4,0)\F6=JOINT(1,0,6,0)\F7=JOINT(1,0,7,0)\F8=JOINT(1,0,8,0),C=RForce X
    ↪ Y Z RTorque X Y Z
!
!--------------------------- ANALYSIS SETTINGS ---------------------------
!
OUTPUT/
, REQSAVE
!, GRSAVE
!
!RESULTS/
!, XRF
!
END
```

B ChParserAdams documentation

We list here some of the more important functions in the `ChParserAdams` class. For more details, see the Project Chrono API documentation [5].

Parse the specified ADAMS input file and create the model in the given system.

```
void chrono::utils::ChParserAdams::Parse(
ChSystem& system,
const std::string& filename
)
```

Arguments:
system containing Chrono system
filename `adm` input file name

Parse the specified ADAMS input file and create the model in a new system. Note that the created system is not deleted in the parser's destructor; rather, ownership is transferred to the caller.

```
ChSystem* chrono::utils::ChParserAdams::Parse(
const std::string& filename,
ChMaterialSurface::ContactMethod method = ChMaterialSurface::NSC
)
```

Arguments:

filename `adm` input file name

method contact method (NSC: non-smooth, complementarity-based; SMC: smooth, penalty-based)

Set body visualization type.

```
void SetVisualizationType(VisType val)
```

Arguments:

val visualization mode (default: NONE)

The visualization mode can be one of:

LOADED use visualization assets loaded from the `adm` file (currently limited to BOX, ELLIPSOID, or CYLINDER)

NONE no visualization

Obtain a reference to the parser's report object.

```
const Report& GetReport() const
```

Arguments: N/A

Print parser report to `stdout`.

```
void PrintReport() const
```

Arguments: N/A

References

[1] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In

T. Kozubek, editor, *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, pages 19–49. Springer, 2016.

[2] MSC Software. ADAMS. `http://www.mscsoftware.com/product/adams`. Accessed: 2015-02-07.

[3] Will Estes. FLEX: The Fast Lexical Analyzer - scanner generator for lexing in C and C++. `https://github.com/westes/flex`. Accessed: 2018-01-03.

[4] Project Chrono Development Team. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `https://github.com/projectchrono/chrono`. Accessed: 2017-05-07.

[5] Project Chrono. ProjectChrono API Web Page. `http://api.projectchrono.org/`. Accessed: 2017-10-20.