LA-UR-16-23235
Approved for public release; distribution is unlimited.

Title: DIORAMA Location Type User's Guide

Author(s): Terry, James Russell

Intended for: Report

Issued: 2016-05-09

DIORAMA Location Type User's Guide

J. Russell Terry

29 January 2015

Contents

1 Introduction

The purpose of this report is to present the current design and implementation of the DIORAMA location type object (LocationType) and to provide examples and use cases. The LocationType object is included in the diorama-app package in the diorama::types namespace.

Abstractly, the object is intended to capture the full time history of the location of an object or reference point. For example, a location may be specified as a near-Earth orbit in terms of a two-line element set, in which case the location type is capable of propagating the orbit both forward and backward in time to provide a location for any given time. Alternatively, the location may be specified as a fixed set of geodetic coordinates (latitude, longitude, and altitude), in which case the geodetic location of the object is expected to remain constant for all time.

From an implementation perspective, the location type is defined as a union of multiple independent objects defined in the DIORAMA tle library. Types presently included in the union are listed and

described in subsections below, and all conversions or transformation between these location types are handled by utilities provided by the tle library with the exception of the "special-values" location type.

1.1 Special-valued location type

The special-valued location object is the only sub-type of the LocationType union that is not implemented in the tle library. This object is implemented as a member class of the LocationType object called LocationType::SpecialValue. This object is intended to handle specialized locations and trajectories. However, at present, an unspecified location is the only special-value location type that has been implemented. This special value is intended to indicate an unspecified location and serves as the default type of the LocationType union if no parameters are provided to the object constructor.

1.2 The geodetic location type (PosGeoWGS84)

The geodetic location object within the DIORAMA LocationType union is defined as a PosGeo class template (provided by the tle library) with a geoid of WGS84 and reference frame of ITRF.

1.3 The Earth-centered, Earth-fixed location type (PosITRF)

The earth-centered, earth-fixed Cartesian location object within the DIORAMA LocationType union is defined as a PosXYZ class template (provided by the tle library) with a reference frame of ITRF.

1.4 The Earth-centered, inertial location type (PosTEME)

The earth-centered, inertial Cartesian location object within the DIORAMA LocationType union is defined as a PosXYZ class template (provided by the tle library) with a reference frame of TEME.

1.5 The two-line element set location type (TLE_orbit_WGS72)

The near-Earth orbit location type within the DIORAMA LocationType union is defined as the TLE_orbit_WGS72 type definition, which is provided by the tle library. Within the tle library, the propagation of a two-line element set is handled by the SpacetrackOrbit object, which allows the user to choose a gravity model (geoid) for the propagation. However, in the DIORAMA API, emphasis has been placed on specifying choices such as geoid and reference frame at compile time within object types. Therefore, a class template called TLE_orbit has been defined in the tle library that is a near-trivial subclass of the SpacetrackOrbit object. The TLE_orbit class template has a single template parameter, which specifies the geoid to use for orbit propagation, and the primary difference between the TLE_orbit template and the underlying SpacetrackOrbit object is that the geoid is taken from the template parameter and cannot be altered by the user. The tle library provides a type definition called TLE_orbit_WGS72, which consists of a TLE_orbit object with a

WGS72 geoid. The WGS72 geoid is used as the default for TLE orbit propagation in DIORAMA because this is the gravity model used by NORAD to produce TLE sets for tracked bodies in orbit.

2 API development and reference

The DIORAMA location type (diorama::types::LocationType) is a simple class consisting of a single member data object: the location variant. The location variant type is a union of multiple position and trajectory types provided by the tle library. However, since the C++ union specifier is restricted to plain-old-data types (e.g., int, double, etc.), the union is constructed using the BOOST variant class template. The list of member sub-types of the variant are given in Section 1 above, and one may review the API documentation for the tle library (and the diorama-app library in the case of the SpecialValue sub-type) for further details on the design and implementation of these objects.

The interface for the location type is fairly lean, only implementing methods that existed prior to a refactoring to incorporate the variant member datum. The LocationType object includes a default constructor that creates a location variant with an "unspecified location" special value sub-type. The destructor is trivially implemented, and the interface includes dedicated constructors for each of the following sub-types: TLE, geodetic, Earth-centered/Earth-fixed Cartesian, and Earth-centered/inertial Cartesian. There is no constructor for the special value sub-type since, at present, the only defined special type is "unspecified location", which is used by the default LocationType constructor. Given the prolific use of constant references in the DIORAMA API, at present, all explictly defined accessor methods for the LocationType object are defined with the const specifier, and the only means of modifying a LocationType object is by assignment from another LocationType object.

Accessor methods of the LocationType object are briefly discussed here. For a more complete (and possibly up-to-date) documentation of the LocationType object and supporting sub-type objects, one should review API documentation for the diorama-app and tle packages. The LocationType object provides member methods Geo() and XYZ() to respectively convert the location to a geodetic or Cartesian Earth-centered, Earth-fixed position. Note that there is no method for converting the LocationType object to an inertial position (or orbit state vector). The Geo() and XYZ() methods provide a single tle::time::gps_time_t argument. In the case that the LocationType object has an Earth-fixed sub-type (i.e., kLocationWGS84 and kLocationITRF), a time argument is not required for these methods. However, for inertial and TLE sub-types, a time argument must be provided; otherwise, a runtime error is thrown.

The LocationType provides an "equal-to" comparison operator for comparison between location types. This method was included to better facilitate unit testing. This comparison operator only returns true if the sub-types of each LocationType are identical and the sub-type objects are equivalent. The LocationType object also provides access to the enumerated sub-type identifier through the Type() methods and a STL string representation of the location through the GetLocationString() method.

As the LocationType object is intended to be a more abstract interface for a set of position and trajectory objects, for the most part member data specific to a given sub-type is typically not accessible through the LocationType interface. However, since propagation of TLE orbits is a

common task in DIORAMA, an accessor method to get the epoch for the orbit (GetEpoch) is included in the LocationType API. This method only returns a valid tle::time::gps_time_t for a TLE sub-typed location type. In all other cases, a special value of "not_a_date_time" is returned.

Given that the LocationType object does not provide any significant access to methods and parameters of the underlying sub-types, the API includes a method for casting the variant to its sub-type object type. This is achieved using the GetAs method template, and example use cases are given in section 3 below.

3 Use cases

3.1 Instantiating a LocationType object

To create a TLE location type follow the example given in listing 1.

Listing 1: Instantiating a DIORAMA LocationType object with TLE sub-type.

```
std::vector<std::string> TLE = {
  "GPS BIIA-10 (PRN 32)    ",
  "1 20959U 90103A    14288.52961711 -.00000043  00000-0  00000+0 0  5389",
  "2 20959  54.2811 204.1626 0114410 355.3978    4.5268  2.00568263174942"
};
diorama::types::LocationType location(TLE);
```

To create a geodetic location type follow the example given in listing 2.

Listing 2: Instantiating a DIORAMA LocationType object with geodetic sub-type.

```
auto altitude  = 2.E4 * datur::units::km;
auto latitude  = 0. * boost::units::degree::degrees;
auto longitude = 0. * boost::units::degree::degrees;
diorama::types::LocationType location(tle::PosGeoWGS84(latitude,longitude,x));
```

To create an Earth-centered, Earth-fixed Cartesian location type follow the example given in listing 3.

Listing 3: Instantiating a DIORAMA LocationType object with Earth-centered, Earth-fixed Cartesian sub-type.

```
auto x = 2.E4 * datur::units::km;
auto y = 0.   * datur::units::km;
auto z = 0.   * datur::units::km;
diorama::types::LocationType location(tle::PosITRF(x,y,z));
```

To create an Earth-centered, inertial Cartesian location type follow the example given in listing 4.

Listing 4: Instantiating a DIORAMA LocationType object with Earth-centered, inertial Cartesian sub-type.

```
auto x = 2.E4 * datur::units::km;
auto y = 0.   * datur::units::km;
auto z = 0.   * datur::units::km;
diorama::types::LocationType location(tle::PosTEME(x,y,z));
```

Although it is doubtful that the user would need or want to do this, for completeness, to create an unspecified location type follow the example given in listing 5.

Listing 5: Instantiating a DIORAMA LocationType object with an unspecified location sub-type.

```
diorama::types::LocationType location;
```

3.2 Assignment of a location type object

Presently, the LocationType assignment operator is trivially implemented by the compiler. There is no implementation of specialized assignment operators for specific sub-types of the LocationType. Therefore, assignment to a LocationType object can only be made from another LocationType object. For example, the example code given in listing 6 assigns a TLE location type to a default-constructed LocationType object.

Listing 6: Instantiating a DIORAMA LocationType object with Earth-centered, sub-type.

```
// instantiate a location type
diorama::types::LocationType location;
// the default-constructed location object has a 'special-value' location sub-type

// assign a TLE location to the default constructed object
std::vector<std::string> TLE = {
  "GPS BIIA-10 (PRN 32)     ",
  "1 20959U 90103A    14288.52961711  -.00000043  00000-0  00000+0 0  5389",
  "2 20959  54.2811 204.1626 0114410 355.3978    4.5268  2.00568263174942"
};
location = diorama::types::LocationType(TLE);
// the location object now has a TLE location sub-type
```

3.3 Obtaining Earth-fixed positions for a DIORAMA location type

Presently, the LocationType object only provides methods for obtaining Earth-centered, Earth-fixed positions, both geodetic and Cartesian using respectively the Geo and XYZ object methods. Each of these methods takes a single GPS time argument (tle::time::gps_time_t). If the time argument is not provided, a default special value of "not_a_date_time" is assumed as a default argument. Examples of obtaining Earth-fixed positions from a location object of various sub-types are given in listing 7. The case of obtaining positions from a TLE sub-typed location are treated in section 3.4 as this is also the method for propagating a TLE orbit in DIORAMA.

Listing 7: Obtaining Earth-centered, Earth-fixed positions from a DIORAMA location.

```
// create an Earth-centered, inertial location
auto x = 2.E4 * datur::units::km;
auto y = 0.   * datur::units::km;
auto z = 0.   * datur::units::km;
diorama::types::LocationType inertial_location(tle::PosTEME(x,y,z));

// convert inertial position to Earth-fixed geodetic at given time
tle::time::gps_time_t time_0 = tle::time::FromUTCString("2014-01-01T00:00:00.0Z");
tle::PosGeoWGS84 geo_0 = inertial_location.Geo(time_0);

// create an Earth-centered, Earth-fixed Cartesian position
diorama::types::LocationType earth_fixed_location(tle::PosITRF(x,y,z));

// convert the Earth-fixed position to geodetic. note that a time argument is not
// required. if an argument is given, it is quietly ignored.
tle::PosGeoWGS84 geo_pos = earth_fixed_location.Geo()
```

3.4 Progating a TLE location type

The propagation of a TLE orbit is a common task within the DIORAMA framework. The TLE sub-typed location has been developed to handle orbit propagation under the hood such that no input from the user is required other than specification of the two-line element set. Therefore, propagation of a TLE orbit is simply a special case of obtaining positions from a location type. Calls to the underlying TLE orbit propagation library are handled as-needed behind the scenes. An example of propagation of a TLE orbit in DIORAMA is given in listing 8.

Listing 8: Propagating a TLE orbit in DIORAMA.

```
// create a TLE sub-typed location object
std::vector<std::string> TLE = {
  "GPS BIIA-10 (PRN 32)     ",
  "1 20959U 90103A    14288.52961711 -.00000043  00000-0  00000+0 0  5389",
  "2 20959  54.2811 204.1626 0114410 355.3978    4.5268  2.00568263174942"
};
diorama::types::LocationType location(TLE);

// get the epoch from the location
// note that this method only returns a valid date and time for a TLE sub-typed
// location. all other sub-types return a not_a_date_time special value
tle::time::gps_time_t epoch = location->GetEpoch();

// get the geodetic and Cartesian position of the TLE location type at epoch
tle::PosGeoWGS84 geo_at_epoch = location.Geo(epoch);
tle::PosITRF     xyz_at_epoch = location.XYZ(epoch);

// get the geodetic position of the orbit at epoch + 1hr
tle::time::gps_time_t TLE_epoch_plus_1hr = TLE_epoch + 60.*datur::units::minute;
tle::PosGeoWGS84 get_at_epoch_plus_1hr = location.Geo(epoch_plus_1hr);
```

3.5 Obtaining the sub-type identifier of a location object

The location type can behave differently with respect to execution of object methods, depending on the underlying location sub-type. Therefore, it will often be necessary to test for a given sub-type before executing a block of code. The example code given in listing 9 shows how to access the location sub-type identifier and to test for specific sub-types.

Listing 9: Identifying DIORAMA location sub-type.

```
// assume that the object 'location' has been previous declared as a
// diorama::types::LocationType
if (location.Type() == diorama::types::kLocationTLE) {
  // execute code for TLE sub-typed location
} else if (location.Type() == diorama::types::kLocationWGS84) {
  // execute code for geodetic sub-typed location
} else {
  // other location sub-type identifiers include:
  //   kLocationSpecial,
  //   kLocationITRF,
  //   kLocationTEME, and
  //   kLocationSpecial
}
```

3.6 Casting a location type to its underlying sub-type

The DIORAMA location type serves as a unified interface for a number of fixed position and trajectory objects. Many of the specialized features or parameters of the various location sub-types are not made available through the DIORAMA location object. Therefore, the location type provides a method for "casting" the location to an object of the underlying sub-type type[1]. It is important to note that the location object can only be cast to an object type that matches its underlying sub-type. For example, a TLE location type can only be cast to a tle::TLE_orbit_WGS72-typed object, and a Earthed-centered, Earth-fixed Cartesian location can only be cast to a tle::PosITRF-typed object. Example code for location casting based on sub-type is given in listing 10.

[1]The term "casting" is quoted here because, as opposed to a standard C++ casting operator, the boost::get function is used to convert the location type member data to the underlying sub-type.

Listing 10: Casting DIORAMA location by sub-type.

```
std::vector<std::string> TLE = {
  "GPS BIIA-10 (PRN 32)    ",
  "1 20959U 90103A   14288.52961711  -.00000043  00000-0  00000+0 0  5389",
  "2 20959  54.2811 204.1626 0114410 355.3978   4.5268  2.00568263174942"
};
diorama::types::LocationType location(TLE);

if (location.Type() == kLocationTLE) {
  auto tle_orbit = location.GetAs<tle::TLE_orbit_WGS72>();
  // execute code with tle TLE_orbit object here
} else if (location.Type() == kLocationWGS84) {
  auto geo_pos = location.GetAs<tle::GeoPosWGS84>();
  // execute code with tle geodetic position object here
}
```

3.7 Obtaining a string representation of a location object

The DIORAMA location type object provides an STL string of simplified location information through the LocationType::GetLocationString method. This method is used to fill status and error messages throughout the DIORAMA code. An example use case for this method is shown in listing 11.

Listing 11: Example using DIORAMA location string method.

```
// assume location is an object of type diorama::types::LocationType
try {
  // calling Geo method with no time argument. throws an exception if
  // the location sub-type is not geodetic or Earth-fixed Cartesian
  auto geo_pos = location.Geo();
  // execute code with geo_pos object here
} catch (std::exception& e) {
  std::cerr << "Caught exception ---" << e.what() << std::endl;
  // provide a little more information if we are debugging
  if (verbosity >= VERB_debug) {
    std::cerr << location.GetLocationString() << std::endl;
  }
}
```