Standards Council

Blockchain in Transport Alliance Standards Council (BiTAS)

BiTAS Tracking Data Framework Profile

REVISION HISTORY

Version	Date	Author
Tracking Data Framework Profile	February 27, 2019	Ben Kothari, BiTAS Tracking Data Work Group Chair

ACKNOWLEDGEMENTS

CORE CONTRIBUTORS: <u>BiTAS Tracking Document Work Group</u>

Ben Kothari, WG chair	Ampliflex
Doug Anderson	Kleinschmidt Inc.
June Arnold	BNSF
Riley Banks	V&S Midwest Carriers Corporation
Mathew Bigelow	Ziing Final Mile Inc.
Sebastian Cochinescu	SupplyBlockchain, Inc.
Subhash Chowdary	Aankhen Inc.
Emmanuel Eriksson	Stratelogics Software Inc.
Trever Erlich	Kenco Logistics
Alan Fong	Fleet Complete
Stacey Geringer	Collabrian Design & Technology
Maneesh Grover	Wipro Technologies
Elizabeth Henderson	Purolator Inc.
Mike Johnson	Don Hummer Trucking
Jim Masloski	Customs Direct
Jim McCullen	Century Distribution Systems, Inc.
Robert Petti	Lading Corporation
Krzysztof Radecki	DAC SA
Devin Reilly	Custom Pro Logistics
Rick Ryan	Pitney Bowes
Jon Van Winkle	VTV Solutions
Tim Wilson	FedEx Services

APPROVERS: <u>2019 BiTAS Board of Directors</u>

Dale Chrystie, chair	FedEx Services
Bart Boudreaux	BNSF
Ken Craig	McLeod Software
Mike Dieter	Transplace
Scott Friesen	Echo Global Logistics
Craig Harper	JB Hunt
Steve Hausman	Triumph Business Capital
Dan Heinen	Kleinschmdit
Lori Heino-Royer	Daimler
Vaneeta Johnson	Delta Cargo
Mark Kessler, Sr.	Trimble Transportation Group
Kevin Martin	Freightwaves
Mauricio Paredes	P&S Transportation
Gil Perez	SAP
Mahesh Saharanaman	UPS
Brad Taylor	Omnitracs
Amihai Zeltzer	Salesforce

COPYRIGHT

CONVENTIONS

The Blockchain examples cited in this Specification are stored on a JSON string. An example of this format is:

```
trackableEvent1 = {
  "ID" : "trackableEvent1",
  "name" : "Schedule Pickup",
  "location" : "location123", //link to location where freight will be picked from
  "trackableEntityID" : "trackableEntity123",  // link to entity which is being tracked
  "reportedByID" : "party123",
  "reportedByRole" : "Shipper",
  "participants" : [
     {
       "ID" : "party456",
       "role" : "Carrier"  // Service Provider responsible for moving the freight
     },
     {
       "ID" : "party789",// Operations Supervisor who is handing over freight to Carrier
       "role" : "OperationSupervisor"
     }
  ],
  "performedTime" : "2019-02-11T23:27:57-08:00"
}
```

Figures and charts included in this specification are drafted using Unified Modelling Language (UML) class diagrams and charts. For more information on how to read the charts and diagrams in this Specification, please refer to Section 6: Component Model.

Contents

1.0 Foreword

This document presents a model to track events which are generated by various processes within a transportation network. This is a framework profile developed by the Blockchain In Transport Alliance Standards Council (BiTAS) and, as such, several terms and definitions related to tracking are introduced in this document. Significant effort has been spent defining these terms.

Moreover, how these terms are related to each other to enable tracking lays the foundation work for several other blockchain Work Groups in the BiTAS standards developing organization. These terms include Shipment, ShipUnit, HandlingUnit, Conveyor, TrackableEntity, Meta Class, MeasureType and References.

It is critical that readers of this profile, including other blockchain Work Group leads, clearly understand the definition of these terms and how they relate to each other. Definitions and requirements defined in this framework profile impact other BiTAS Specifications, ensuring that terms and their meanings are consistent across BiTAS Specifications.

2.0 Introduction

The purpose of the BiTAS Tracking Data Framework Profile is to define core data structure and *links* to other data structures required to answer the basic question – “Where is my Shipment?” The purpose is to provide adequate information to quickly determine the location of a shipment and to see how it is progressing through its planned route and operational processes until it reaches its final delivery destination. Shipment tracking is a very broad topic involving many processes, participants, systems, sensors and software applications. This Profile is primarily focused on *Location* tracking among participants within a transportation network. Other aspects of tracking may be covered in future releases.

The core data structures, *TrackableEvent*, *TrackableEntity, EventParticipant, MeasureType, Dimension* and *Reference* are introduced in this document. This Profile describes the data format, cardinality and relationship between these data structures. In addition, it defines how these core data structures reference Location, Shipment and Party data structures, which are being defined by other BiTAS work groups.

Figure 2 below shows an example where status information transmitted from devices (financial and operational (*finOps*) applications, etc.) may be codified in a TrackableEvent data structure and stored within Blockchain in near chronological order. Transportation Management Systems (TMS), portals and apps may *query* TrackableEvent data stored within Blockchain ledger to determine the current location and status of the shipment.

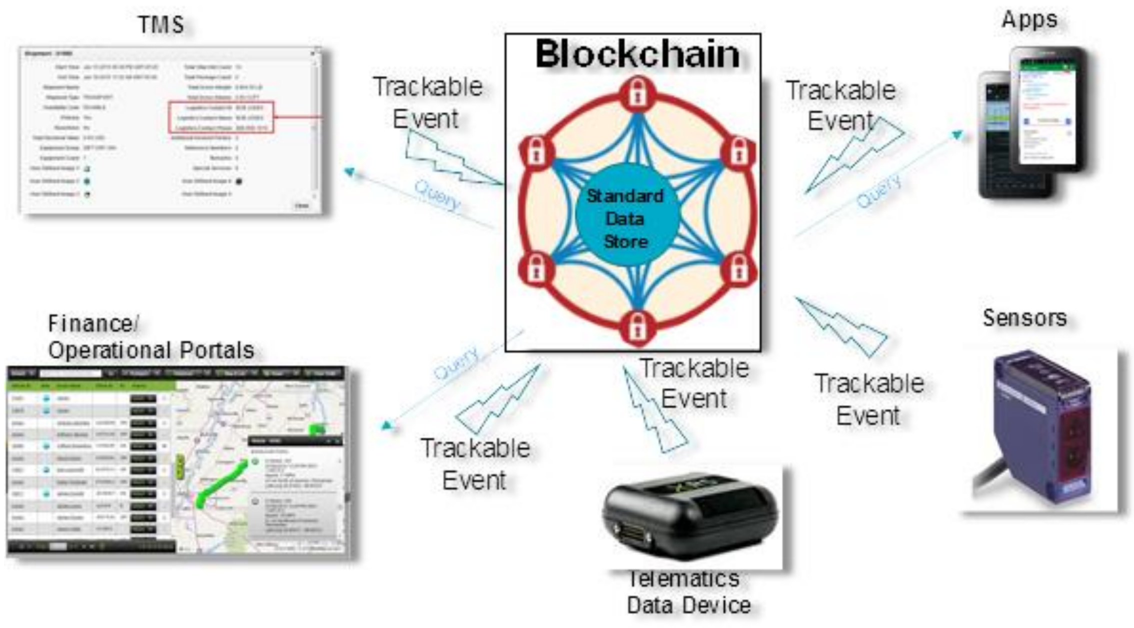


Figure 2: Examples of Events being generated by various systems, applications and devices

3.0 Scope

The current scope of this Profile is to only define and provide Location tracking information for Shipment entities. In the future, this Profile may be enhanced to include custodial, environmental and security related information. The primary focus of this Profile is to define data structures related to how data may be persisted in a storage system including Blockchain Ledger. This data may be queried by participants to find the current location status of Shipment entities. Messaging data formats and other IoT streaming data formats are currently out of scope.

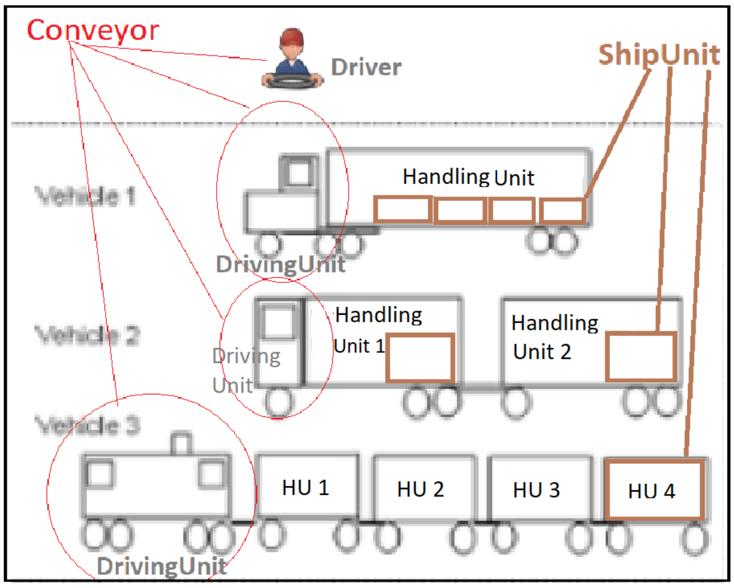
4.0 Normative references

Open GIS / IOT references as detailed in Section 11 of this Specification: External identifier of this OGC® document: http://www.opengis.net/doc/is/sensorthings/1.0

5.0 Terms, Abbreviations and Definitions

For the purposes of this document, the following terms and definitions apply:

Term or Abbreviation	Definition
Shipment	Groups of orders may be combined by a shipper. A set of orders to be moved from/to a location on a given date time by any conveyor is referred to as *Shipment*. Multiple orders can be grouped into a single Shipment, or a Shipment may include items from different orders and may include backorders as well as regular orders. Shipment represents the movement of freight from one location to another, activity at a cross-dock, and non-moving charges associated with shipping activity. A shipment is uniquely identified by an ID, and stores information related to total weights, but the tracking occurs at the individual ShipUnit level.
ShipUnit	A Shipment may contain multiple *ShipUnits (SU)* which may be tracked separately or as one shipment entity for certain segments of the transportation network. A Shipment is a logical collection and aggregation of one or many ShipUnits. Each ShipUnit is uniquely identified and tracked, but all share the same associated Shipment's ID (ShipmentID). ShipUnit describes how freight is packaged for transportation. ShipUnit is the lowest form possible for a *TrackableEntity* (defined below). ShipUnit inherits from TrackableEntity, which is a recursive data structure. However, ShipUnit is not recursive.
Conveyor	Conveyor is any vehicle or Party responsible for moving the Shipments.

	Active vehicle resources such as tractors or locomotives are collectively referred to as *DrivingUnit (DU).* "Motor Engine" or a power generating portion of the vehicle, which is responsible for moving Shipments, is an example of a DrivingUnit. Telematics devices and sensors are usually installed in a DrivingUnit. A driver and the DrivingUnit he is operating is collectively referred to as a Conveyor. In a simplest case, a person walking up a flight of stairs to hand-deliver a box is also a Conveyor.
HandlingUnit	Passive vehicle resources which are responsible for bearing or storing the load of ShipUnits are collectively referred to as *HandlingUnit (HU).* The smallest loadable unit of a vehicle which is used to transport goods such as a container is an example of HandlingUnit. Boxes, parcels, pallets, envelopes, etc. are examples of HandlingUnit. ShipUnits (defined above) are packed and carried inside a HandlingUnit.  Conveyor Driver ShipUnit Vehicle 1 Handling Unit DrivingUnit Vehicle 2 Handling Unit 1 Driving Unit Handling Unit 2 Vehicle 3 HU 1 HU 2 HU 3 HU 4 DrivingUnit *Figure 5: Picture showing HandlingUnit, ShipUnit, DrivingUnit, Conveyor* Each HU may be tracked separately, or an HU may contain multiple HUs which may be tracked together as a single entity. The capacity of the equipment is defined in terms of Effective Weight, Effective Volume, Equipment Reference Units (ERUs) and equipment dimensions, Compartments, Length, Width and Height. HandlingUnit inherits from *TrackableEntity* (defined below), a recursive data structure. HandlingUnit is a *recursive* representation of HandlingUnit itself, but is specific to EquipmentType.

	Definition of Shipment, ShipUnit and HandlingUnit is expected to be refined further in Specifications from the BiTAS Shipment Component Work Group.
Party	Party is any Enterprise (Company, Corp, LLC, etc.) that participates and provides services within a transportation network. Party may also represent any individual person, or group of people who participate within the network. A driver who operates a truck is an example of a Party. Party will be fully defined by the BiTAS Party Component Work Group.
Resource	Any human, device or machine responsible for transporting ShipUnits, which needs to be tracked, is a Resource.
Document	Document is defined to be any document(s) required to track and monitor the execution of instructions or agreements which are assigned to Shipments or Equipment.
Link	Link is a unique identifier within a data structure record which refers to (or points to) to another record stored elsewhere in the Blockchain ledger. Within database terminology, the purpose of link is like a foreign key relation between two tables. In the UML model diagram, a dotted line between attributes of two classes is shown as Link.
TrackableEvent	TrackableEvent class is defined to store the information related to any event which may happen during the journey of a shipment. It contains sufficient information such that, upon querying, a user will be able to track the current location and status of a shipment.
TrackableEntity	TrackableEntity is a genesis class for anything trackable within the blockchain. Currently, it serves as the base class for ShipUnit, HandlingUnit, Document, and/or Conveyor. HandlingUnit is further extended to create several subclasses including ShipUnit, HandlingUnit, Trailer, Pallets, Boxes, Envelopes, etc. In the future, it may be used by other BiTAS Work Groups to define additional sub-classes or objects which may need to be tracked in the Blockchain. Every time an event is written in the Blockchain, a TrackableEvent object is created which represents the details about the event. Each TrackableEvent in the Blockchain links to TrackableEntity via *trackableEntityID*. TrackableEntity includes an attribute *parentEntityID* which points to itself. The recursive feature of TrackableEntity enables modelling of hierarchical grouping of related things. For example, a Box class inherits from TrackableEnity, a Pallet class inherits from

	TrackableEntity and also a Trailer class inherits from TrackableEntity. The parentEntityID of Box points to Pallet and parentEntityId of Pallet points to Trailer, and so on. This allows tracking information right from the box to the Trailer in the HandlingUnit Hierarchy. TrackableEntity contains an array named "contents" which holds IDs of TrackableEntity or its subclasses. IDs stored in "contents" array in the blockchain refer to SU and HU objects which need to be tracked in the Blockchain.

6.0 Component Model

Data structures defined within BiTAS working groups are represented as UML class diagrams. Each data structure is defined as an object-oriented *Class*. A class may inherit common attributes from a parent class. Common *Meta* class defines the common attributes which are required for each class defined by any work group. Objects shown in blue color are core objects required for tracking. Classes shown in yellow are classes related to Shipment and Party and will be further extended by those work groups; however, the relationship between yellow and blue classes may not be changed. Any changes to relationship between classes will change how tracking is accomplished. A detailed example is shown below to illustrate how tracking is accomplished by using this model.

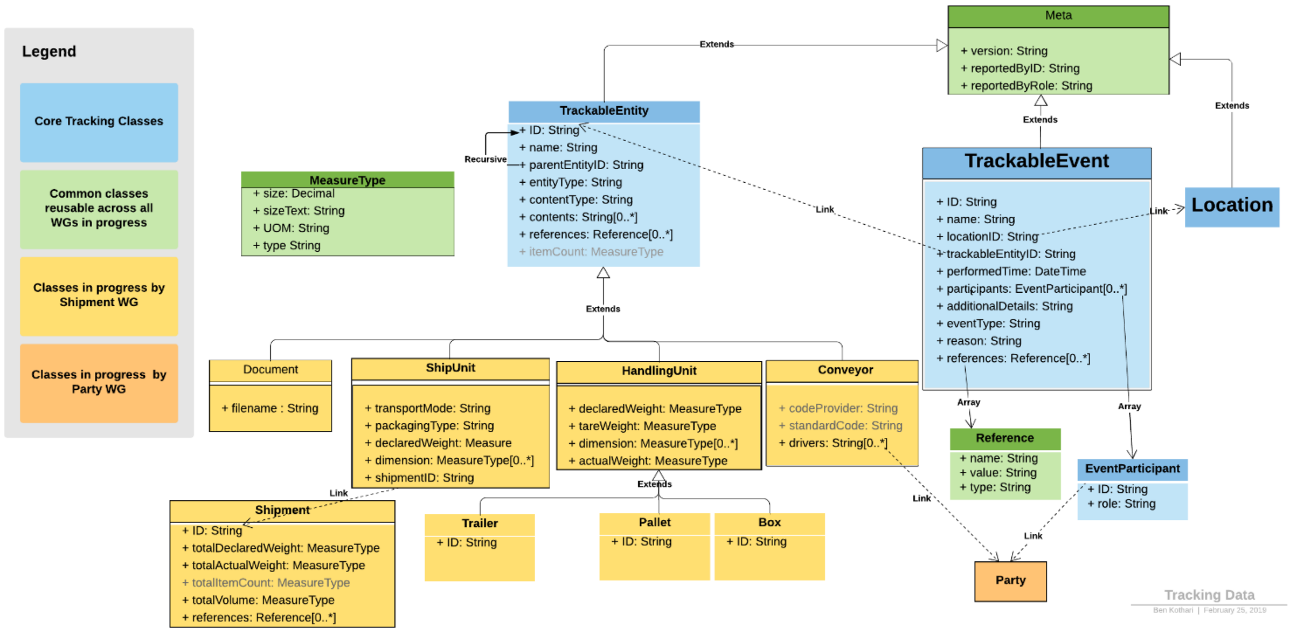


Figure 6: Tracking Entity Model

7.0 Common Classes

7.1 Meta Class

The purpose of Meta class is to define common attributes which are needed by all entities across various Work Groups. To avoid duplication of the same information within each class, a separate BiTAS Work Group will be responsible for defining common attributes. Meta class in figure 5 is shown in a gray color. It is shown as an example to illustrate the requirements to a separate Work Group.

Attribute Name	DataType	Purpose
Meta	Class	
version	String	current version of the document
reportedByID	String	Party ID responsible for writing data to the blockchain ledger is already provided blockchain framework. reportedByID represents on whose behalf this data was written.
reportedByRole	String	Role of the reporting party.

7.2 MeasureType Class

To represent 5 pounds of weight, users may use measure.size = 5 and measure.UOM=lbs.

Attribute Name	DataType	Purpose
MeasureType	Class	
size	Decimal	any decimal value
sizeText	String	any alphanumeric value
UOM	String	Unit of Measure. E.g. lbs, tones

7.3 References

References are an embedded array within TrackableEvent to store any IDs or information of other entities to better corelate events. The purpose of this information is to match events with other Documents or Entities with the Blockchain as well as external systems. This class may also be by Shipment and Party Work Groups as an extensibility mechanism.

Attribute Name	Attribute DataType	Purpose
Reference	Class	any name-value pair
name	String	E.g. WayBill, BOL#
value	String	E.g. 34379789
type	String	Here are various types of references. E.g. Tracking Number

8.0 Tracking Classes

8.1 TrackableEvent Class

The table below list the attribute names, datatypes and the purpose, required in order to build TrackableEvent data structure.

Attribute Name	Attribute DataType	Mandatory	Purpose
Standards			
ID	String	Yes	Unique identifier for the event.
name	String	Yes	Unique name which defines the purpose of the event. Name and meaning of each event will be provided as example list.
locationID	String	No	Location where the event is generated or originated from.
trackableEntityID	String	Yes	Entity which is being tracked. Entity may be referred to as ShipUnit, HandlingUnit, Document, etc. This attribute only stores a link to the class being referenced. The actual referenced class is stored elsewhere on the blockchain.
performedTime	String	Yes	Actual time when the event is performed. This time may be different from the time when information is stored in the Blockchain ledger.
participants	EventParticipant [0..*]	No	List of participants (their unique identifier and party role) participating in this event. This is represented as an embedded array within TrackableEvent.
eventCategory	String	No	This attribute may be used to filter events based on certain categories (performance optimization). There may be further subcategorization of events as more use cases are discussed. Broadly, event should be categorized as normal or unexpected.
reason	String	No	In case of an unexpected or exception event, report the reason for this event.

TrackableEntity	Class	Yes	Broadly, TrackableEntity is used to define ShipUnit, various types of HandlingUnits, Conveyor and/or Document. Our recommendation is to use TrackableEntity as a parent class to store common attributes for each trackable entity which may be defined within your Work Group. Currently, this class is used within the context of Shipment Tracking, but it is generic in nature, and may be used to track additional assets in the future.
references	Reference [0..*]	No	References are an embedded array within TrackableEvent to store any IDs or information of other entities to better correlate events. References may also be used with other entities such as ShipUnit, Equipment etc. However, when they are used within the context of TrackableEvent, they usually refer to the information associated with this event.

8.2 EventParticipant class

This is represented as an embedded array within TrackableEvent. This class shows the information related to who issued the event, and the list of participants who are the recipient or impacted by this event.

Attribute Name	**DataType**	**Mandatory**	**Purpose**
EventParticipant	**Class**		
ID	String		points to Party Class ID
role	String		Party roles examples would be Driver, ShipmentPlanner, OperationSupervisor, etc.

8.3 TrackableEntity

TrackableEntity is a genesis class for anything trackable within the blockchain. Currently, it serves as a base class for ShipUnit, HandlingUnit, Document, and/or Conveyor. HandlingUnit is further extended to create several subclasses including Trailer, Pallets, Boxes, Envelopes, etc. In the future, it may be used by other Work Groups to define additional sub-classes which may need to be tracked. TrackableEntity contains an attribute,” *parentEntityID*”, which holds the ID of a parent TrackableEntity.

For example, a Box class inherits from TrackableEnity, a Pallet class inherits from TrackableEntity and also a Trailer class inherits from TrackableEntity. The parentEntityID of Box points to Pallet and parentEntityId of Pallet points to Trailer, and so on. This allows tracing information right from the box to the Trailer in the HandlingUnit Hierarchy.

TrackableEntity contains an array named “contents”, which holds IDs of TrackableEntity or its subclasses. IDs stored in “contents” array in blockchain refer to SU and HU objects which need to be tracked in Blockchain.

Attribute Name	**DataType**	**Purpose**
TrackableEntity	**Class**	
name	String	Name of the TrackableEntity.
parentEntityID	String	ID to TrackableEntity which serves as the parent or container to the current entity. This enables recursive structure.
entityType	String	Currently supported types are ShipUnit, various types of HandlingUnits, Conveyor, Document.
contentType	String [0..*]	Type of contents. This may include commodity type. Hazard classification, etc..
contents	String [0..*]	IDs of content which are being tracked in Blockchain. entityType defines the type of content which may be tracked, and this array stores their IDs.
references	Reference [0..*]	any additional references, e.g. master tracking number associated with this entity.
itemCount	MeasureType	this attribute is shown grayed out. This attribute may not be needed as the number of content IDs may be derived by querying the size of the array.

9.0 Example

The diagram in Figure 9 shows an example of how Shipment may be tracked by the logical model defined in this Profile. Consider the following scenario:

1. Each iphone sold in stores would be a Packaged Item.
2. Six iphones (Packaged Item) go into a small box. The small box would have packagingType. packagingType = 'OneRate Box' specifies the type of carton.
3. Two hundred small boxes go onto a pallet. The pallet would be of class Handling Unit – Pallet as defined in the model above.
4. Twenty pallets fit into a twenty-foot container. The container would be of class Handling Unit - Trailer (type: Container).
5. Four Containers are loaded on 1 Handling Unit - Trailer. Same class is used to represent both Handling Units (HU), Trailer and Container here.

Specification Step:
First, you must define a ShipUnit (SU) Specification for small box of 6 iPhones. Then assign packagingType of 'OneRate Box' to it. Shipment class stores “totals” information related to its collection of SUs. Each SU’s shipmentID points to Shipment class.

Consolidation Step:
Depending on your scenario, you may skip consolidation step and directly track SUs. In this case *contents* array in TrackableEntity will store list of all the SU’s ID. However, in our scenario we are showing consolidation step where several SUs are assigned to multiple levels of HUs.

During consolidation step,

- SUs will be assigned to *HU – Pallet*. SU’s parentEntityID will point to *HU – Pallet*.
- HU-Pallet’s parentEntityID will be assigned to *HU - Trailer (type: Container),*
- and then container will be assigned to *HU – Trailer* via its parentEntityID.

The above consolidation is one illustration, many other combinations are possible. For example, if it's pallets, then it's associated to a trailer directly; if it's parcels, then it could be in a container which is then loaded in a trailer; or a bulk loaded trailer full of parcels would have the SUs directly associated to the trailer.

Tracking:
Now a TrackingEvent named "Tendering" is created on the Blockchain which points to a TrackableEntity (TE) via trackableEntityID. Depending on how much information you would like to store and share on the Blockchain, TE's *contents* array attribute may be assigned. If you are interested in only tracking the outermost HU, which in our case would be of type HU-Trailer, then contents array will contain IDs of HU - Trailer. Or if you want to share and track all levels of information, then contents array may to points SUs which will require you to store all SUs and HUs on the blockchain. parentEntityID of SU will point to HU-Pallet and HU-Pallet's parentEntityID will point to HU- Trailer, and so on. This enables model to track entities at any level.

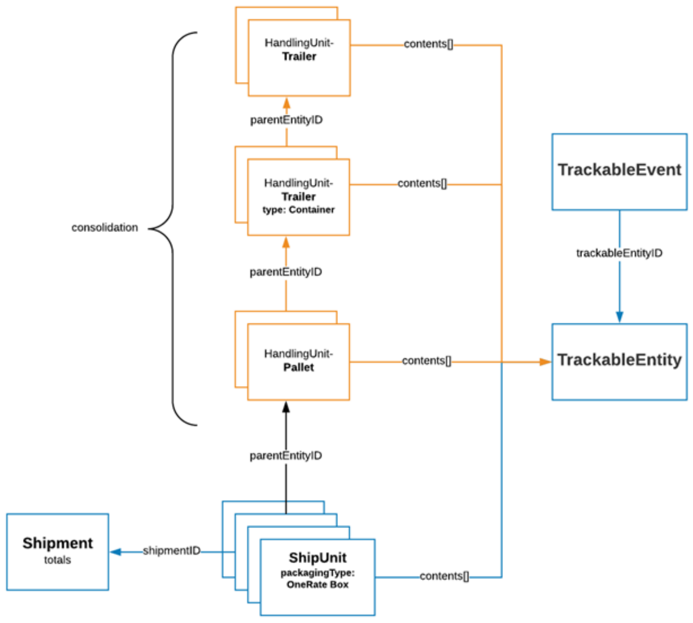


Figure 9: Example showing how shipment may be tracked

9.1 Simple Box Tracking

Figure 9.1 below shows the simplest tracking example where only 1 ShipUnit (SU) is created and loaded on the Blockchain first, and then TrackableEvent (Evt) is created. trackableEntityID attribute of TrackableEvent links to this SU.

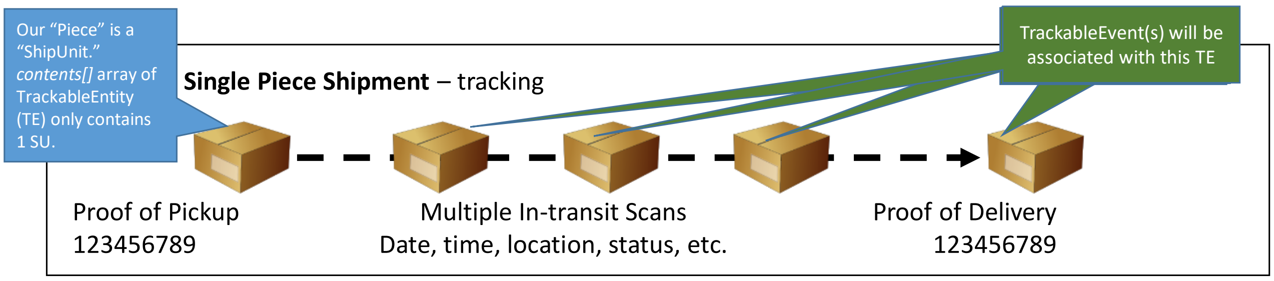
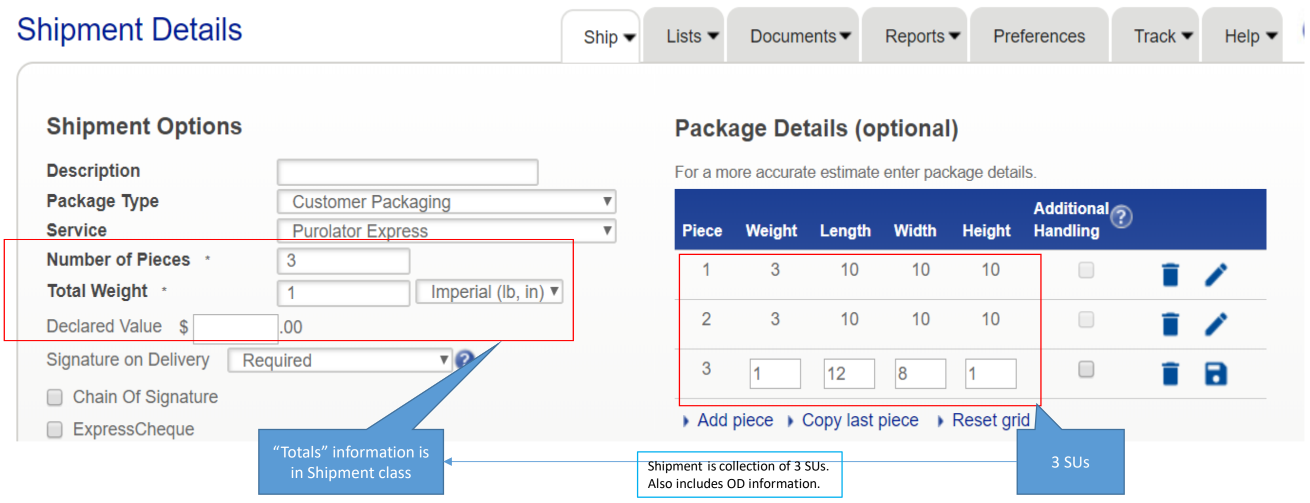

Our "Piece" is a "ShipUnit." *contents[]* array of TrackableEntity (TE) only contains 1 SU.
Single Piece Shipment – tracking
TrackableEvent(s) will be associated with this TE
Proof of Pickup
123456789
Multiple In-transit Scans
Date, time, location, status, etc.
Proof of Delivery
123456789

Figure 9.1: Simple example of tracking 1 SU

9.2 Multiple Package Tracking

The screen below shows a shipment being created with 2 boxes and 1 packet. All 3 entities will be modelled as SUs and tracked together as 1 TrackableEntity.



Shipment Details

Shipment Options

Description	
Package Type	Customer Packaging
Service	Purolator Express
Number of Pieces *	3
Total Weight *	1 Imperial (lb, in)
Declared Value $	.00
Signature on Delivery	Required

Chain Of Signature

ExpressCheque

Package Details (optional)

For a more accurate estimate enter package details.

Piece	Weight	Length	Width	Height	Additional Handling
1	3	10	10	10	
2	3	10	10	10	
3	1	12	8	1	

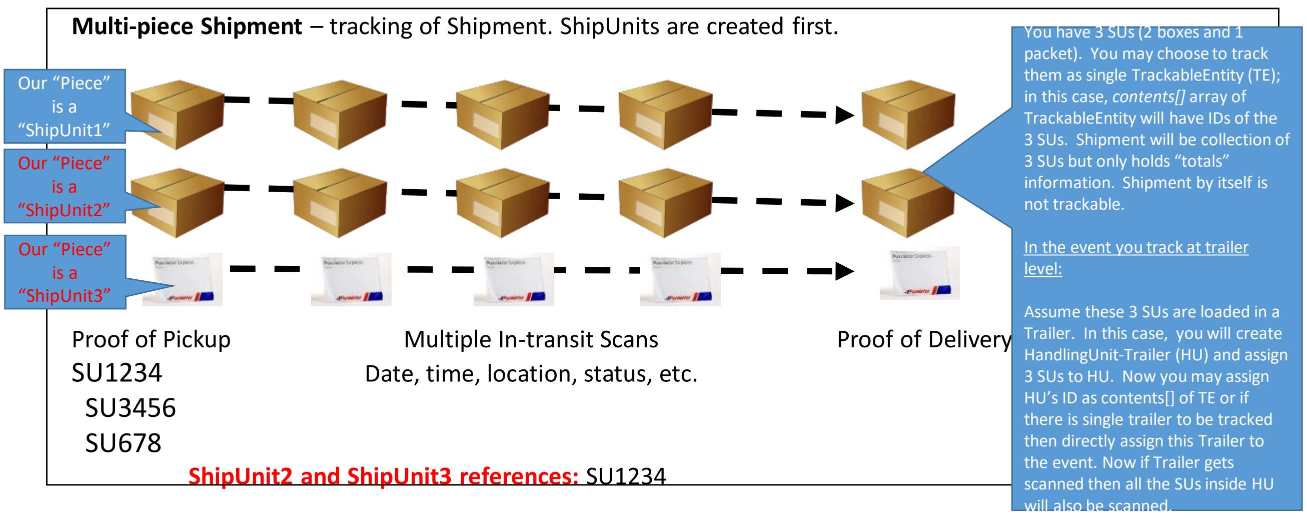
"Totals" information is in Shipment class

Shipment is collection of 3 SUs. Also includes OD information.

3 SUs

After the shipment is created, in this case, three shipping labels would be created and one would be attached to each piece. The PIN number on the first label is the Shipment PIN.

Figure 9.2a: Example User Interface showing creation of Shipment with 3 SUs


Our "Piece" is a "ShipUnit1"
Our "Piece" is a "ShipUnit2"
Our "Piece" is a "ShipUnit3"

Multi-piece Shipment – tracking of Shipment. ShipUnits are created first.

Proof of Pickup
SU1234
SU3456
SU678

Multiple In-transit Scans
Date, time, location, status, etc.

Proof of Delivery

ShipUnit2 and ShipUnit3 references: SU1234

You have 3 SUs (2 boxes and 1 packet). You may choose to track them as single TrackableEntity (TE); in this case, *contents[]* array of TrackableEntity will have IDs of the 3 SUs. Shipment will be collection of 3 SUs but only holds "totals" information. Shipment by itself is not trackable.

In the event you track at trailer level:

Assume these 3 SUs are loaded in a Trailer. In this case, you will create HandlingUnit-Trailer (HU) and assign 3 SUs to HU. Now you may assign HU's ID as contents[] of TE or if there is single trailer to be tracked then directly assign this Trailer to the event. Now if Trailer gets scanned then all the SUs inside HU will also be scanned.

Figure 9.2b: Example showing how SU, TE are assigned

In the example above, 3 SUs are created on the Blockchain and their IDs are assigned to contents array of 1 TrackableEntity. Then an event "Schedule Pickup" is created and the trackableEntityID points to this TE.

JSON example illustrates how the model is created for tracking purpose:

```
// Example 1:
// A Shipper is scheduling a pickup for 2 boxes of iPhones and 1 packet of manual from a Carrier to be
shipped to Best Buy.

trackableEvent1 = {
   "ID" : "trackableEvent1",
   "name" : "Schedule Pickup",
   "location" : "location123", //link to location where freight will be picked from
   "trackableEntityID" : "trackableEntity123",  // link to entity which is being tracked in defined below.
   "reportedByID" : "party123",
   "reportedByRole" : "Shipper",
   "participants" : [
      {
         "ID" : "party456",
         "role" : "Carrier"  // Service Provider responsible for moving the freight
      },
      {
         "ID" : "party789",// Dock Manager who is handing over freight to Carrier
         "role" : "DockManager"
      }
   ],
   "performedTime" : "2019-02-11T23:27:57-08:00"
}

// 1 trackableEntity is tracking 3 ShipUnits (SU) as a collection
trackableEntity123 = {
   "ID" : "trackableEntity123",
   "name" : "Best Buy iPhone Shipment",
   "contentType" : "ShipUnits",
   "contents" : ["SU1234", "SU3456", "SU678"]
}

SU1234 = {
   "ID" : "SU1234",
   "transportMode" : "LTL",
   "packagingType" : "BOX",
   "contentType" : "iPhone XR",
   "entityType" : "ShipUnit",
   "declaredWeight": {
      "size" : "120.0",
      "UOM" : "LBS"
   }
}

SU3456 = {
    "ID" : "SU3456",
    "transportMode" : "LTL",
    "packagingType" : "BOX",
    "contentType" : "iPhone XR",
    "entityType" : "ShipUnit",
    "declaredWeight": {
      "size" : "120.0",
      "UOM" : "LBS"
    }
}

SU678 = {
   "ID" : "SU678",
   "contentType" : "iPhone XR",
   "transportMode" : "LTL",
   "packagingType" : "PACKET",
   "entityType" : "ShipUnit",
   "declaredWeight" : {
      "size" : "7.5",
      "UOM" : "LBS"
   }
}

// First you will create 3 SUs, then create 1 TrackableEntity (TE) and assign contents' array of with 3 SU's ID,
// and then create TrackableEvent (EVT)
```

9.3 Multiple level Tracking

Extending the above example further, assume the SUs are loaded on a Pallet and then the Pallet is a loaded on a Trailer. In this case, you will create HandlingUnit-Pallet (HU) and assign 3 SUs shown above to this HU. Then Pallet will be assigned to HU-Trailer. JSON example below shows how relations are created for an event named “Arrival at Source”.

```


// Example 2:
// More complex example showing hierarchy of SU and HU.  Shipper has packed SUs on HandlingUnit - Pallets,
// and Trailer from Carrier is being loaded with Shipper's Pallets

trackableEvent2 =  {
   "ID" : "trackableEvent2",
   "name": "Arrival at Source",
   "location": "location456",
   "trackableEntityID": "trailer1234",
   "reportedByID": "party123",
   "reportedByRole": "Carrier",
   "participants" : [
      {
         "ID" : "party123",
         "role" : "Shipper"   // Shipper whose pallets are being loaded
      },
      {
         "ID" : "party789",   // Dock Manager responsible for handing over freight to Carrier
         "role" : "DockManager"
      }
   ],
   "performedTime": "2019-02-11T23:27:57-08:00"

}

trailer1234 = {
   "ID" : "trailer1234",
   "entityType": "Trailer",
   "transportMode" : "TL",
   "actualWeight": {
      "size": "2000",
      "UOM": "LBS"
   }
}

// Following entities are created by Shipper
HU7000 = {
  "ID" : "HU7000",
  "name": "Pallet",
  "transportMode" : "LTL",
  "parentEntityID" : "trailer1234",  // attribute inherited from TrackableEntity Class, points to
Carrier's trailer1234
  "actualWeight": {
      "size": "200",
      "UOM": "LBS"
   }
}

SU7001 = {
  "ID" : "SU7001",
  "name": "ShipUnit1",
  "transportMode" : "LTL",
  "parentEntityID" : "HU7000",  // attribute inherited from TrackableEntity Class, points to Pallet
  "declaredWeight" : {
      "size" : "170.5",
     "UOM" : "LBS"
   }
}
```

```
SU7002 = {
  "ID" : "SU7002",
  "name": "ShipUnit2",
  "transportMode" : "LTL",
  "parentEntityID" : "HU7000", // attribute inherited from TrackableEntity Class, points to Pallet
  "declaredWeight" : {
     "size" : "200",
     "UOM" : "LBS"
   }
}
```

10.0 Event Categorization

Broadly, Event may be categorized as planned or unplanned. Any Event which is expected or occurs during normal processing of shipment may be categorized as planned. Categorization of Events may be required in order to filter the number of Events returned within a Blockchain ledger query.

10.1 Planned Events

A list of planned events is proposed in the embedded spreadsheet as an example. The Tracking Data Work Group is releasing the logical model for review and a list of events will be published as part of future work.

A list of Event names and their definitions is included in the spreadsheet embedded as example here.

Worksheet%20in%2
0BiTAS%20Tracking%

10.2 Unplanned Events

List of exception events may include:

Delay
Damage
Sighting

Pending
Clearance
Overdue
Completed
after Due
Date
Quantity
Change
Due Date
Change
Retender

11.0 Future Versions

As consensus is achieved on standard methodologies for incorporating lists and attributes into this Specification, future versions will be made available detailing these lists and attributes. Event names and Party roles may be predefined in future version. As new standards emerge from parallel workgroups, the definitions that are described here, that are not directly related to tracking will be superseded by those respective groups