Published October 1, 2003. Distribution restricted to Sponsors until January 1, 2004.

AUTO-ID CENTRE

WHITE PAPER

EPC™ Information Service – Data Model and Queries

Mark Harrison

AUTO-ID CENTRE INSTITUTE FOR MANUFACTURING, UNIVERSITY OF CAMBRIDGE, MILL LANE, CAMBRIDGE, CB2 1RX, UNITED KINGDOM

ABSTRACT

The Electronic Product Code (EPC™) is the globally unique object ID in the Auto-ID infrastructure. It serves as a database lookup key in order to access information about a tagged object. An EPC™ Information Service (EPCIS) provides a standard interface for access and persistent storage of EPC-related data, for read and write access by authorized parties. The EPC™ will often be used as one of the parameters in a query sent to an EPCIS, although additional parameters such as a timestamp range or readerID or a particular well-defined attribute of the tagged object (e.g. date of manufacture) are also valid parameters. An EPCIS is not an omniscient entity which attempts to pull together and provide all available information about an EPC™ and return this as a comprehensive PML data packet. Rather, EPCIS is a interface, which handles very specific simple queries efficiently and just returns the relevant data required by that query. It is the responsibility of the client or application to make the required iterative queries to an EPCIS (or multiple EPCIS services across a supply chain) in order to deal with more complex queries. With this approach in mind, this paper examines the different fundamental categories of data which one might want to access via an EPCIS, then proposes three fundamental categories of simple query, which allow more complex queries to be broken down into tractable simple queries which an EPCIS could answer.

WHITE PAPER

EPC™ Information Service – Data Model and Queries

Biographies

Mark Harrison
Senior Research Associate

Mark Harrison is a Senior Research Associate at the Auto-ID Centre lab in Cambridge working on the development of the EPC™ information service (EPCIS), as well as web-based graphical control interfaces and manufacturing recipe transformation ideas. In 1995, after completing his PhD research at the Cavendish Laboratory, University of Cambridge on the spectroscopy of semiconducting polymers, Mark continued to study these materials further while a Research Fellow at St. John's College, Cambridge and during 18 months at the Philipps University, Marburg, Germany. In April 1999, he returned to Cambridge, where he has worked for three years as a software engineer for Cambridge Advanced Electronics/Internet-Extra, developing internet applications for collaborative working, infrastructure for a data synchronisation service and various automated web navigation/capture tools. He has also developed intranet applications for his former research group in the Physics department and for an EU R&D network on flat panel displays.

WHITE PAPER

EPC™ Information Service – Data Model and Queries

Contents

1. INTRODUCTION

In order to promote widespread deployment of radio-frequency identification (RFID) technology, the Auto-ID Centre has focused on storing just the minimal amount of information on a tag, namely a unique identification number for the tagged object. This number (currently represented as just 64 or 96 binary digits or bits) is known as the electronic product code (EPC™) [1] and serves as a lookup key for retrieving additional data about the object from a networked database [2], in a similar manner to the way in which we all use URLs [3] or web addresses as a pointer to further pages of information on the world wide web. This reduces the amount of memory capacity required by the tags, thus lowering their production costs [4]. The Auto-ID Centre also believes in developing open standards for interoperable technology solutions which span the entire supply chain, including suppliers, manufacturers, distributors and retailers. A key component is therefore the networked database interface, through which the additional data about the tagged object can be accessed. This is called an EPC™ Information Service (EPCIS), though previous papers [5,6] have referred to it as a PML Server or PML Service. Although a number of initial prototypes have been developed, the EPCIS specification is still being written. This paper attempts to dispel the notion of the 'big database in the sky' and address the realities, such as the fact that the data will be fragmented across the supply chain and that each company may host their own EPCIS service for co-operation with their trading partners. The Object Name Service (ONS) [7] allows conversion of an EPC™ into an address of at least one EPCIS service on the supply chain, although in addition to this, a sequence registry or dynamic component of ONS has also been proposed [6] to facilitate identification of all relevant EPCIS services within each supply chain, particularly to assist with rapid tracking and trace, as well as product recalls.

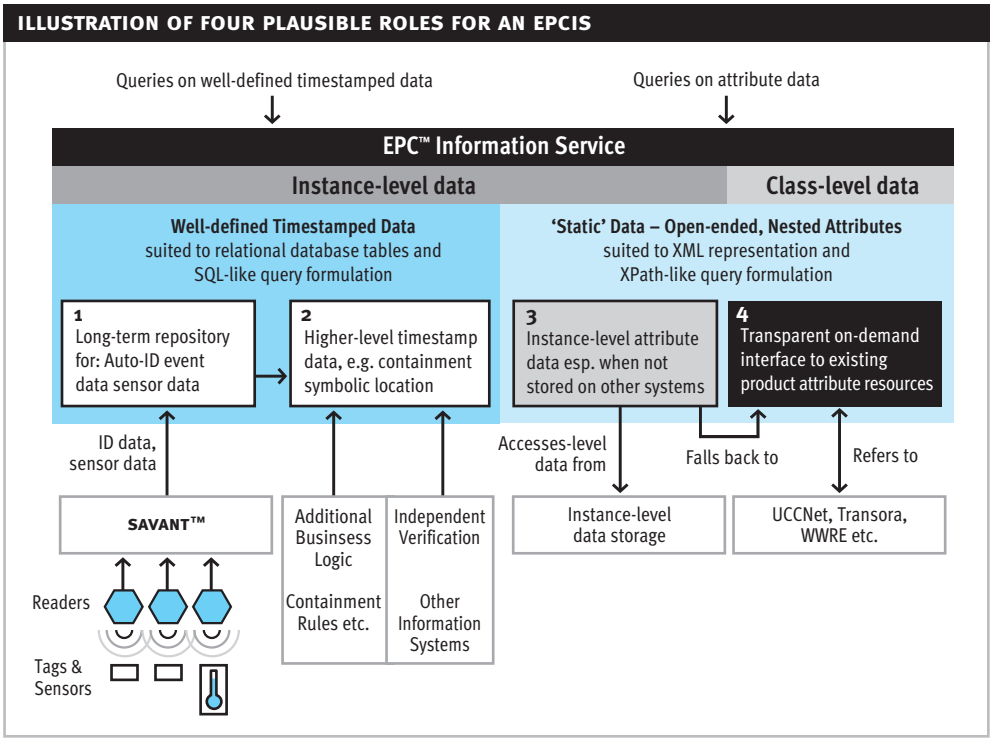
In this paper, we examine the categories of data which may be accessible via an EPCIS and illustrate how an EPCIS can be used to answer real queries driven by use cases, by way of implementation examples, which admittedly draw upon existing query standards such as SQL [8] and XPath [9] for illustrative purposes, although this is not intended to imply a particular favoured storage mechanism for persisting the data, although different categories of data may naturally be more efficiently stored or queried using one approach over another. Having said that, there is plenty of high quality database software and tools, which are available for free or even as open source, so there is plenty of scope for interoperable EPCIS solutions from multiple developers or vendors.

1.1. Definition

An EPC™ Information Service (EPCIS) provides a standard interface for access and persistent storage of EPC-related data, for read and write access by authorized parties.

1.2. Roles

Figure 1: Potential roles and categories of data and query types for the EPCIS.



The distinguishing aspects of Auto-ID technology are the event data of RFID readings and sensor readings, as well as the instance-level granularity of object identification and correspondingly storage of data about each unique object. Therefore in descending order or priority, the roles of an EPCIS are:

1. To provide access to a long-term repository of low-level Auto-ID event data, by which we mean a set of records (Timestamp, ReaderID, TagEPC) when tags are detected by readers. The reason is that Savant [10] is optimised for rapid collection and efficient filtering of data but does not necessarily provide its own long-term repository for the data. Auto-ID event data also includes sensor measurements as a more general case. RFID readers are a form of identity sensor, where the corresponding value is the identity (EPC™) of the tagged object.

2. To provide access to higher-level timestamped data, such as records of containment or symbolic location, which may in part be derived from the lower-level tag-read event data, combined with other information, such as human verification of packing/unpacking operations or rules expressing containment hierarchies in terms of parent-child relationships expressed using EPC™ patterns.

3. To provide storage and access to those attributes of an object which are defined at instance level (serial number granularity), which may not currently be stored at that level of granularity in existing business information systems, e.g. serial-level temperature history, date and time of manufacture, expiry date.

4. To provide transparent on-demand access to existing product attribute data held in other repositories and data synchronisation systems, such as UCCNet [11], Transora [12], WWRE [13] etc., especially as a fall-back, where the corresponding attribute or property is not defined at instance-level.

2. CATEGORIES OF DATA

EPC-related data falls into two categories, as shown in Table 1:

- timestamped historical data
- static attribute data about an object

Table 1: Two distinct categories of EPC-related data

'EPC-RELATED' DATA	
Timestamped historical data	**Static attribute data**
– Tag readings – Symbolic location / containment – EPC –› Transaction ID	– Attributes defined at serial level – Attributes defined at product level

2.1. Timestamped Historical Data

The timestamped historical data may express a sequence of low-level tag readings, location/containment assignments or may even refer to business transactions which involve the object, such as purchase orders, advance shipping notices etc., by reference to a transaction ID or URL, together with an identification of the issuing entity. Figure 2 illustrates four distinct sub-categories of timestamped data which may be accessed in order to answer queries about EPC-tagged objects. We identify the following:

Observations
- which object was seen, by which reader, when

Transactions
- which objects were associated with a particular transaction
- which transaction is associated with an unexpected object

Measurements
- historical data from sensors
- correlate with observations to deduce temperature etc.

Containment/Symbolic Location
- identifies the enclosing object or the location identity
- not represented as spatial co-ordinates e.g. (x,y,z)

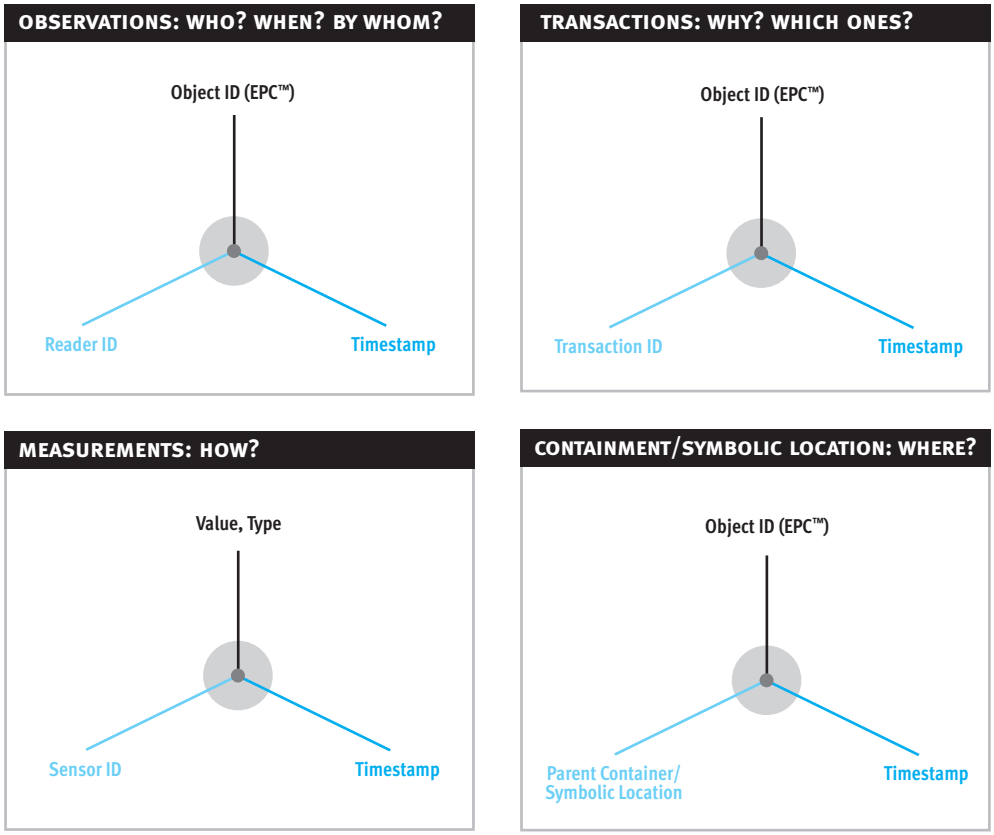


Figure 2: Four sub-categories of timestamped data related to EPC-tagged objects.

It is perhaps worthwhile briefly commenting on the idea of containment or symbolic location. This is shown schematically in Figure 3 as a set of objects connected by parent/child relations. Note that in some examples, the parent can be a physical container which encloses its children, just as a case encloses its items. However, the parent may also be a symbolic location, such as being on a particular forklift truck or in a particular storage bay. The symbolic location is not a direct geographic representation of location, such as (x,y,z) or polar co-ordinates. Rather it is a way of indicating that an object is in/on/at a particular other object or location, even if the other object is mobile or its static location is defined elsewhere in geographic terms.

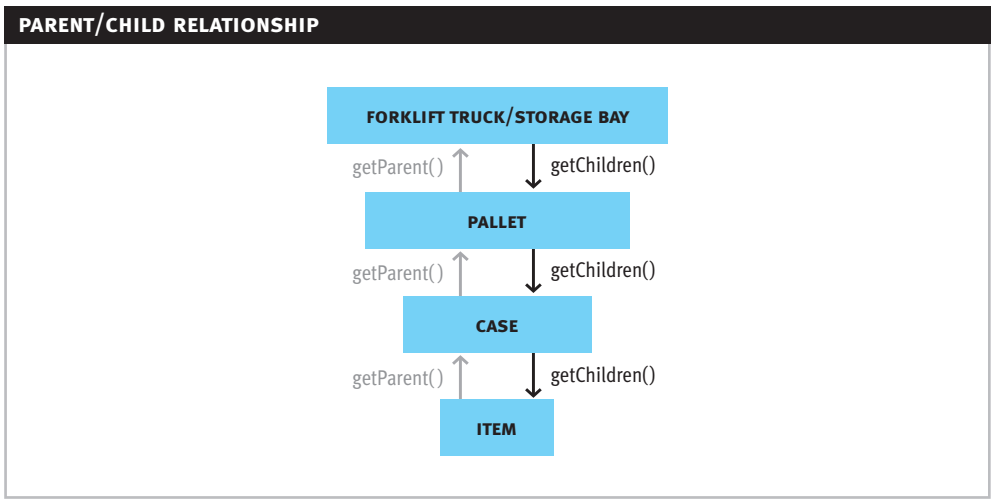


Figure 3: Parent/child relationships and the functions which connect them.

Because of the need to be able to easily traverse the parent-child tree in order to find the entire hierarchy of parents or children or just those parents or children of a particular type, we propose in section 3.0.2 some functions which allow convenient traversal of the parent-child containment data.

We consider that complex queries may be handled by combining data accessed from multiple EPCIS services across a supply chain. While the EPC gives us a globally unique ID value for an object, other ID types such as Sensor ID, Reader ID or Transaction ID may need to be qualified with an indication of ownership or static location in order to make the ID values globally unique across the supply chain.

As several objects pass through a particular company, we collect a number of observations, measurements, and refer to or update details about transactions and known containment or location of those objects. We can view these as points or trajectories in corresponding 3-D spaces, as shown in Figure 4.

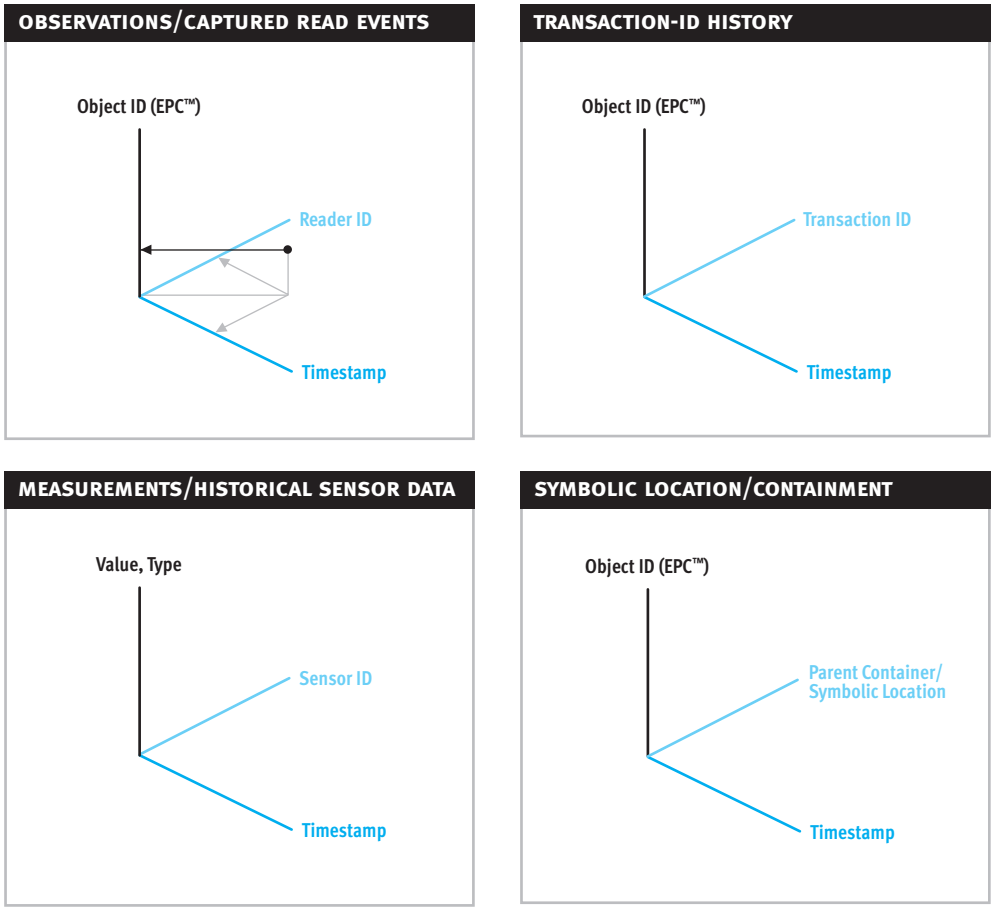


Figure 4: Three-dimensional representation of triples of data points which make up observations, measurements, transactions and records of containment or symbolic location.

In addition to the timestamped data represented in Figure 4, there is information about the readers and sensors themselves, such as their fixed locations or sensor type and units of measure of sensor values, which is in many cases essentially static and could be defined once in simple database tables. This sort of information about the systems infrastructure is very useful for correlating observations with values of physical properties, obtained from measurements. It is also essential if the readings from different types of sensors are stored together simply as numeric values in the same database table. In this way, only the sensors matching a particular context or type (e.g. temperature sensors) would be selected by the query.

2.1.1. Timestamped Data – Implementation in Relational Database Tables

The 3-D representations of timestamped data spaces, shown in Figure 4 could be implemented or persisted as a few simple relational database tables with a limited number of fields or columns, usually including timestamp, EPC™, then a third column such as readerID, parentContainer, transactionID, as illustrated in Figure 5.

Figure 5: Well-defined relational database tables for data from readers and sensors and about containment or symbolic location.

TIMESTAMPED DATA

Reader_data		
Timestamp	Object EPC™	Reader ID

Containment_data		
Timestamp	Object EPC™	Parent EPC™

Sensor_data		
Timestamp	Value	Sensor ID

Transaction_data		
Timestamp	Object EPC™	Transaction ID

Sensor_type		
Sensor ID	Measurement Type	Unit

Transaction_type		
Transaction ID	Transaction Type	URL to host

Note that because of the potential for communication delays or clock synchronisation problems, we may actually want to record two timestamp fields – one corresponding to the time at which the update was received by EPCIS, the other corresponding to the timestamp contained within the data packet received by EPCIS.

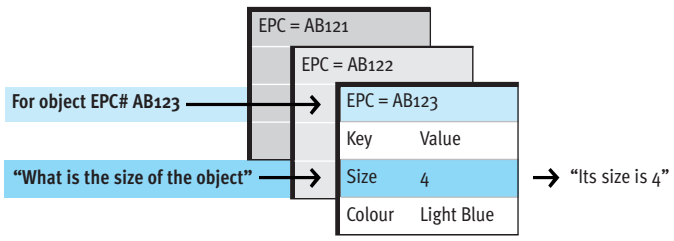
2.2. Static Attribute Data

Static attribute data may contain a much wider range of attributes than the timestamped data discussed in section 2.1. Furthermore, these attributes are likely to be much more specific to particular industry verticals.

At first sight, a simple approach would be to store a simple key-value table of attributes for each EPC™, as illustrated in Figure 6.

Figure 6: Storing a hash table of key-value attribute pairs for each EPC™

KEY-VALUE TABLE OF ATTRIBUTES



However, a better approach is to ensure interoperability between trading partners by ensuring that the lookup key is expressed using well-defined XML [14] schema [15] which have already been standardised for particular industry sectors [16].

In the example of Figure 6, the simple attribute 'size' could mean different things in different contexts, e.g. a shoe size within a numeric country-specific scale of shoe sizes, or the dimensions of a box (length x width x height) in particular units. A schema is chosen which is relevant to a particular context or industry sector and which precisely and unambiguously defines what the attribute refers to.

XML schema express a hierarchical XML markup structure by defining the XML elements and in-line XML attributes which may appear in a particular markup, as well as defining rules for how various elements may be nested within each other and which elements are optional, compulsory or may occur more than once.

For example, for cans of potatoes, we may want to distinguish between the gross weight (including the weight of the can and liquid) and the net weight of just the potatoes contained within the can. Both may be referred to as 'mass' or 'weight' [17] – but in the XML markup, one attribute could be enclosed within a 'Gross dimensions…' container, while the other could be enclosed within a 'Net dimensions…' container, as illustrated in Figure 7. This is the XML approach to putting the names of properties into a context – or fully qualifying them so that everyone can refer to precisely the same property in the same uniform way, overriding differences of language or choice of terminology.

Figure 7: An example of an extensible standards-based approach for storing general attribute data using well-defined XML schema, allowing also for a more hierarchical data model with nested elements, as illustrated by the example XPath expression, which identifies the gross weight property, indicated by the shaded background.

```
<productData>
  <grossDimensions>
    <weight units="kg">0.450</weight>
  </grossDimensions>
  <netDimensions>
    <weight units="kg">0.400</weight>
  </netDimensions>
</productData>
```

example XPath expression: `/productData/grossDimensions/weight/value()`

With this approach, rather than the key being a simple text string such as "size" or "expiry date", the key for accessing a particular property would refer to a particular XML Schema by URI [3] and then to a particular element or attribute within that schema by using a path expression, such as XQL[18] or XPath [9] to identify a particular node within the XML schema. In the example of Figure 7, the XPath expressions might look like

`/productData/grossDimensions/weight/value()` to access the gross weight

AND

`/productData/netDimensions/weight/value()` to access the net weight.

The analogy with the file path which appears in operating systems or web addresses should be apparent – a path merely denotes which containers (filesystem directories or XML elements) one must navigate through to access a particular endpoint (e.g. file or value or element in an XML document).

This approach re-uses existing industry-standard XML schema to allow unambiguous access to particular properties of an object and could be applied across all industry sectors, not just the consumer packaged goods sector. It also allows flexibility of adding new attributes via well-defined published XML schema, much more so than if the EPCIS specification arbitrarily were to define which attributes or properties of objects should be included, from a particular viewpoint, such as the retail supply chain.

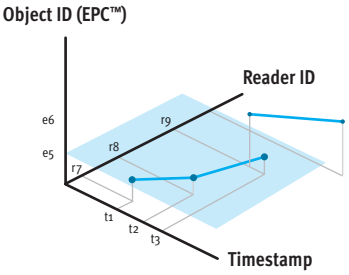
3. QUERY TYPES

We clearly have two distinct categories of data with quite different data structures, namely well-defined tables of timestamped data and trees of essentially static attributes or properties. This is in accord with categories of data generally and the database systems which have been developed to handle them. Relational databases are well suited to handling data which can be expressed in table form with a finite number of fields or column headings. XML tools such as XML::DOM [19], XPath [9]/XQL [18] are better suited to accessing and manipulating tree-like data.

In the case of static attribute data, we are usually wanting to do a simple key-value lookup, as depicted in Figure 6. We look for an exact match of a particular property attribute name, in order to extract a particular element from the XML markup. In this case, an XPath or XQL parameter may be a suitable way of identifying the attribute of interest within a specific XML Schema.

For timestamped data, we may want to pose several different types of query, such as 'which readers saw this tag' or 'which tags were seen by this reader'. Returning to the three-dimensional representation of timestamped data as shown in Figure 4, this is equivalent to looking for points which lie within particular planes (layers) or volumes of the 3-D space, as illustrated in Figures 8, 9, 10.

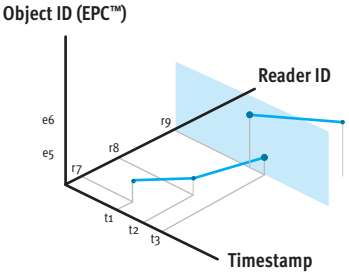
3-D DATA SPACES AND THE USE OF CONSTRAINTS TO ASK DIFFERENT RELATED QUERIES



Object EPC™	Reader ID	Timestamp
e5	r7	t1
e5	r8	t2
e5	r9	t3

Constraint: Object ID = e5

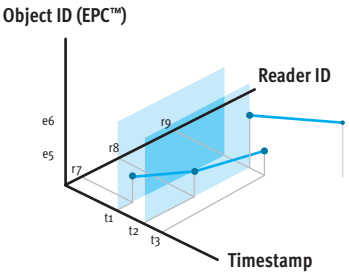
Figure 8: "Tell me all the readers which have seen EPC #e5 and the times they were seen". The shaded plane constrains the EPC™ to a particular value, e5 – and returns just the observations where the EPC™ took that value, together with the corresponding ReaderIDs and timestamps.



Object EPC™	Reader ID	Timestamp
e6	r9	t2
e5	r9	t3

Constraint: Object ID = r9

Figure 9: "Tell me all the objects seen by reader ID # r9 and when they were seen". This time the reader-ID is constrained and just those observations made by reader # r9 are returned, i.e. just those points which lie within the shaded plane.



Object EPC™	Reader ID	Timestamp
e5	r7	t1
e6	r9	t2
e5	r8	t2

Constraint: t1 <= Timestamp <= t2

Figure 10: "Tell me all observations made between times t1 and t2". This time, the timestamp range is the constraint and only those observations made between those timestamps are returned, i.e. the points lying between the two shaded planes.

As we can see, this way of looking at the data allows us to ask several related questions just by formulating the constraints in different ways. Of course it is possible to combine these constraints, to obtain for example just those observations seen by a particular reader within a certain timestamp range.

Within the framework of a relational database approach, such queries are readily formulated using SQL queries of the general form:

```
"SELECT * FROM tablename WHERE constraints_are_true"
```

where constraints express logical conditions on the table columns or fields which must be true if the record or row is to appear in the recordset returned as output from the query. This approach allows a considerable degree of flexibility, including Boolean combinations (AND, OR, NOT) of criteria, numerical range matching, time/date range matching, pattern matching (using substrings or regular expressions). In this section, we realise that we have two markedly different categories of data; on the one hand well-defined timestamped data with a limited number of fields or column headings, on the other a time-independent static document with many arbitrary fields. The virtue of attempting to merge the three query types into a single query method is unclear. We therefore propose three distinct generic query methods, depending on the category of data being queried. Note that the names of the functions are only indicative and may be changed – the purpose of the examples is only to distinguish between the different query approaches and the parameters they might take.

3.1. Queries on Timestamped Data

For timestamped data, we could use the SQL analogy and have a generic method such as

```
GetTimestampedData(tablename,constraints)
```

where tablename may be reader_data, sensor_data, containment_data and constraints is an SQL-format constraint on the fields of the selected table.

3.2. Queries on Containment Data

In addition, we might provide additional convenience methods for the containment/symbolic data to recursively traverse the parent/child relationships shown in Figure 3.

```
GetImmediateParent(epc, timestampconstraint)
GetImmediateChild(epc, timestampconstraint)
```

These would return the immediate parent or child subject to the constraints on the time range. For a more extensive traversal of the parent-child tree of Figure 3, we might provide functions which recursively traverse the tree or containment data table in a zig-zag manner, either returning a hierarchical list of all parents and children (if the EPC™ pattern is set to match all EPCs™) – or alternatively, just filtering out the parent or children which match a particular EPC™ pattern, no matter how many levels away they are in the hierarchy.

```
GetAllMatchingParents(epc, timestampconstraint, epcpattern)
GetAllMatchingChildren(epc, timestampconstraint, epcpattern)
```

3.3. Queries on Static Attribute Data

For static attribute data, we could have a method such as

```
GetStaticData(epc,xml_schema_uri,xpath)
```

where the attribute data is known to be marked up as an XML document for a particular EPC™ and conforming to the specified schema. The XPath expression denotes a particular XML node within that schema, to identify the property of interest.

4. IMPLEMENTATION ISSUES

Although the EPCIS service does not stipulate how the data must be persisted in the underlying databases, it may be more appropriate and efficient to store timestamped data in relational database tables and essentially static data in XML document catalogs. Data binding tools such as Castor [20], JAXB [21] and JDO [22] allow object-relational mapping and object-XML mapping.

4.1. Determining Current 'State' from Historical Data

Timestamped data allows us to determine the historical records or the current state of the system. In terms of RFID reads or location, we use the term 'trace' to look backwards in time at a time-ordered historical list of all the observations of an object, whereas we use the term 'track' to look forward in time and obtain the most recent observation.

This section deals with the particular issues of tracking – or determining the current state or latest observation from a table of historical data. We can pose the question in different ways (or apply different constraints on the fields of the table). For example, we may want the latest observation of a particular tagged object – but it is just as valid that we might instead want the latest observation made by a particular reader. The problem however is that a query about the current state of the system may constrain the reader ID or the tag EPC™, so there is not a well-defined primary lookup key for the table of current 'system state'.

Rather than maintaining separate tables for the current data of each of these fields (i.e. one table of latest objects seen by each reader and another table of the last reader to have seen each object), the following approach allows us to use just two tables:

1. The table of historical log data (as already discussed) for trace purposes. This table is only ever appended – rows are never over-written.
2. An auxiliary table, which just maintains the timestamp of the latest update affecting any of the values within the history table.

By joining these two tables together, we obtain a table which represents the current 'state' of the system and which omits rows which have been wholly superseded by more recent updates.

As an example, consider 3 tags, A,B,C, which are observed by various readers X,Y,Z at timestamps 1-4. The history table may appear as in Table 2:

Table 2: Observation history table for tracking example

TIMESTAMP	TAG	READER
1	A	X
2	A	Y
3	B	X
4	C	Y

If we ask reader Y, 'what was the last tag you read', the answer is 'C'
If we ask tag A, 'which reader last read you', the answer is 'Y'

Just because tag C was read most recently at reader Y, this should not cause the loss or over-writing of the information at timestamp t=2 about the reading of A at Y, since this is the last information we have about tag A.

By timestamp t=4, the 'current state' table would omit row 1 and look like Table 3:

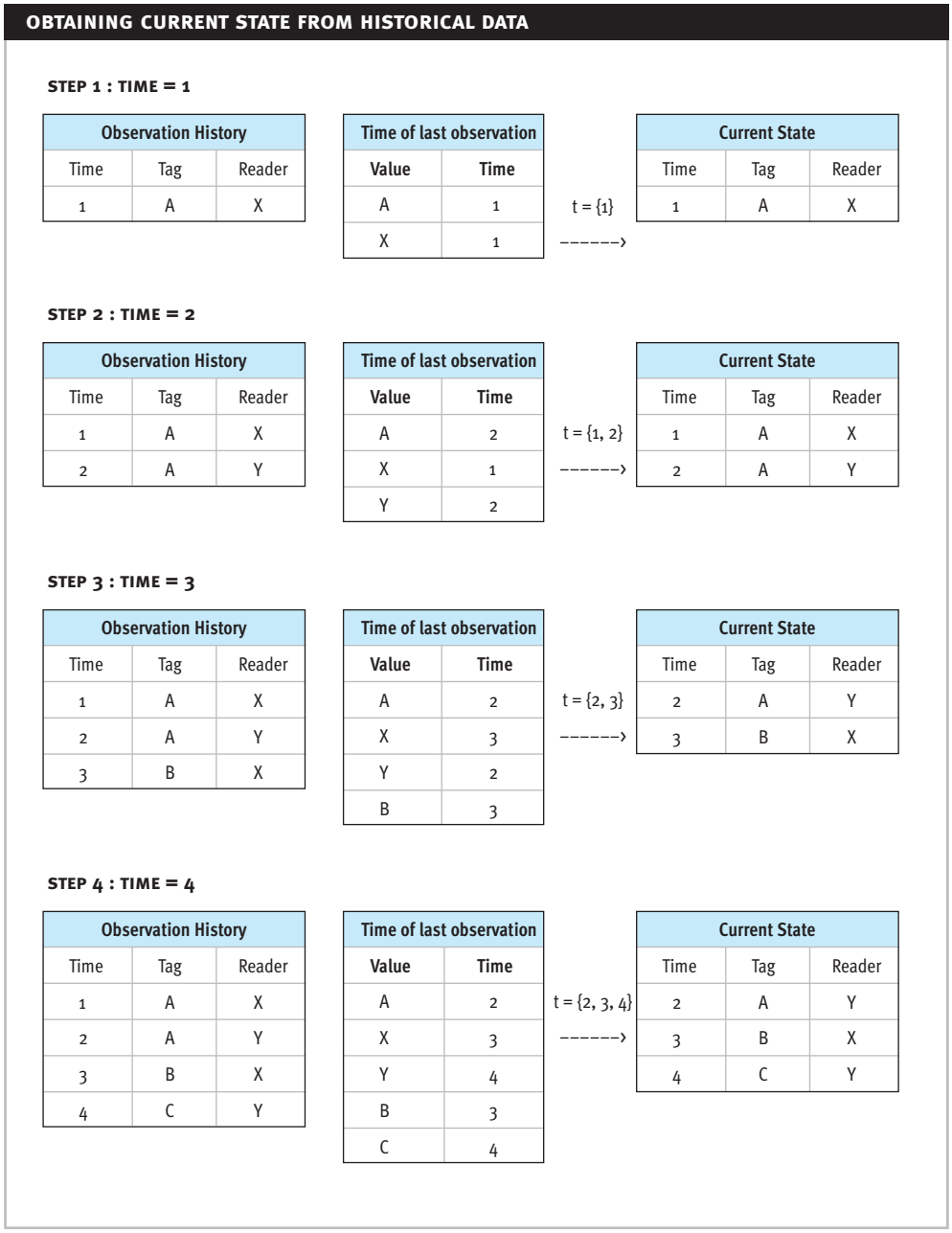
Table 3: Hypothetical 'current state' table for tracking example

TIMESTAMP	TAG	READER
2	A	Y
3	B	X
4	C	Y

The reason why row 1 of Table 2 is absent from Table 3 is that there is new information about BOTH of its tag and reader fields which has superseded it in subsequent rows, i.e. the latest information on tag A is that it was seen by reader Y at timestamp t=2, while the latest information for reader X is that it last saw a tag B at timestamp t=3.

One way of arriving at such a table is to maintain a simple two-column auxiliary table in which the primary key is any value which may be one of the fields other than timestamp. The other column is the timestamp of the latest update which involves that key. e.g. any update which relates to tag A will append the history table and also update the corresponding timestamp in the auxiliary table. The current state is then formed by a join of these two tables where only rows of the history for timestamps in the 'touch' table are returned. This approach may be more scalable than maintaining separate current tables to answer queries such as 'where was tag A was last seen?' and 'which tag was last seen by reader X?'. Figure 11 illustrates this procedure.

Figure 11: Example of how to obtain a table of current 'state' from the timestamped historical data combined with a table which records just the time at which each value (whether a tag or reader) participated in an observation event.



5. EXPRESSION OF SIMPLE QUERIES AND HOW THEY MIGHT BE FORMULATED IN SQL

This section illustrates how many basic queries about observations (triples of tagEPC, readerID and timestamp) could be answered if such data were stored in relational database tables. The solutions are only given as SQL SELECT statements purely by way of example and immediate familiarity for those who develop existing database applications. It does not necessarily imply that SQL will be the actual query language, although it clearly has features which lend themselves to this category of data.

What readers read EPC (x) over time range (t1,t2) ?
SELECT * FROM reader_data WHERE tagEPC = x AND timestamp › t1 AND timestamp ‹ t2

What tags did reader (r) see over time range (t1,t2) ?
SELECT * FROM reader_data WHERE readerID = r AND timestamp › t1 AND timestamp ‹ t2

Which sensors read value › 10°C over time range (t1,t2) ?
SELECT * FROM sensor_data WHERE value › 283 AND timestamp › t1 AND timestamp ‹ t2

What is the temperature history of sensor (s) see over time range (t1,t2) ?
SELECT * FROM sensor_data WHERE sensorID = s AND timestamp › t1 AND timestamp ‹ t2

What EPCs are enclosed in container EPC(x) ?
SELECT * FROM containment_data WHERE parentEPC = x

What are the immediate children of EPC(x) ?
SELECT * FROM containment_data WHERE parentEPC = x

What is the immediate parent of EPC(x) ?
SELECT * FROM containment_data WHERE objectEPC = x

What is the value of sensor (s) at reader (r) over time interval (t1,t2)?
SELECT * FROM sensor_data WHERE sensorID = s AND timestamp › t1 AND timestamp ‹ t2

5.1. Complex Queries Which Might be Considered ‘Simple’

The following examples are more complex in the sense that the questions as expressed in human language are ambiguous – and also the processing of the query may need to draw upon additional information held outside EPC™ network architecture.

What is the object type of EPC(x) ?
either XPath query on attribute data – or may be derived from GTIN-EPC

What are the attributes of EPC(x) ?
Which other XML schema are available on this EPCIS for EPC x besides PML Core?
Retrieve schema. Try querying for attribute required using an appropriate XPath expression.

What is the location of reader (r) ?
- may be available via an XPath query of attribute data keyed on reader ID r
- or may need to access other mapping systems outside the EPC™ network

What sensors are available for reader (r) ?
- may be available via an XPath query of attribute data keyed on reader ID r
- or may need to access other systems

What tags of object type (o) did reader (r) over time interval (t1,t2)?
SELECT * FROM reader_data WHERE readerID = r AND timestamp › t1 AND timestamp ‹ t2
AND tagEPC LIKE ‹epcpattern›
- depends on whether it is easy to specify an EPC pattern for objects of a particular type
- if not, then may need to obtain a list of EPCs™, then check the object type of each via XPath attribute query

Which tags satisfy <logical expression> at reader (r) over time interval (t1,t2)?

SELECT * FROM reader_data WHERE readerID = r AND timestamp > t1 AND timestamp < t2

– then for each EPC™, test logical query on attribute type.

– It may be possible to combine both approaches using something like object query language (OQL), which resembles SQL but instead of returning rows of a recordset, returns collections of objects. In terms of data binding with Castor, the collection may be bound to a relational database table, while the static attribute data could be bound to an XML catalog, keyed on EPC™.

What are the EPCs™ of type (o) at location (l) over time interval (t1,t2)?

Need to specify what is meant by location. Ideally, we want to arrive at a symbolic location, which may be a set of reader EPCs™ – or a containerEPC

SELECT * FROM reader_data WHERE readerID = r AND timestamp > t1 AND timestamp < t2

SELECT * FROM location_data WHERE parentEPC = l AND timestamp > t1 AND timestamp < t2

6. DECOMPOSITION OF (VERY) COMPLEX QUERIES INTO SIMPLE QUERIES

The following three examples are intended to show how a perfectly reasonable question posed by a human end-user would need to be broken down into a sequence of iterative simple queries which the early version EPCIS could handle. If such queries were frequently posed, then it would be appropriate to develop client applications which manage the decomposition of the query and the joins and filtering of data on behalf of the end-user. The results at each stage of the process are indicated by the arrow =>.

6.1. Example: “Where is the nearest red sweater of size 4?”

0. Resolve ambiguity about whether ‘nearest’ actually means ‘at the shortest distance away’ or ‘reachable in the shortest time’

1. Tell me all EPCs stored on local EPCIS subject to static attributes:
   color = red
   size = 4
   AND
   (from real-time inventory / sales data), dynamic attribute:
   already sold = no
   in stock = yes
   => list of EPCs or EPC classes, {e}

2. For each of these EPCs, tell me all readers which have seen these in the last hour
   SELECT * FROM reader_data WHERE tagEPC = e
   => list of readerEPCs, {r}

3. For each of these readerEPCs, tell me your current (x,y,z) location
   XPath query on static attribute data for each readerEPC r
   => hash-table mapping an (x,y,z) for each readerEPC r

4. For each of these (x,y,z) co-ordinates, sort in order of increasing distance from me at (xo,yo,zo)
   ORDER BY (x-xo)*(x-xo) + (y-yo)*(y-yo) + (z-zo)*(z-zo) ASC
   => ordered list of readerEPCs (r)

5. For the nearest n readers r, request static attributes:
   (human-readable) location description
   => "Aisle 5; shelf 3 from top, Aisle 7; shelf 2 from top"

6.2. Example: "Where are the 5 CDs that were supposed to be in the last order?"

1. Find transactionID and issuing entity for 'last order'
   => TransactionID t

2. For transactionID t, what were the EPCs associated with it?
   SELECT * FROM transaction_data WHERE transactionID = t
   => Set of EPCs {e} at serial number granularity

3. Obtain delivery time of goods corresponding to transactionID t
   lookup in other business information system?
   => Time interval (t1,t2), dock-door readerID r

4. Obtain list of EPCs received in time range t1-t2 at reader r
   SELECT * FROM reader_data WHERE Timestamp > t1 AND Timestamp < t2 AND ReaderID=r
   => Set of EPCs {e'} received as the order

5. Find EPCs in dispatch advice / ASN {e}, which are absent from set of objects received {e'}
   => List of missing objects {e"}

6. For each missing EPC in {e"}, use the sequence registry (dynamic ONS component) to track
   to find current custodian
   => Hash-table lookup of custodian address for each EPC in set {e"}

7. Query the EPCIS of each respective custodian for the trace / current location of the EPC
   SELECT * FROM location_data WHERE tagEPC = e"
   => Timestamped list of locations

8. Query the EPCIS of each respective custodian for contact details
   => Telephone number, fax number

6.3. Example: "This case of meat is tainted. Where has it been and which other products crossed paths with it?"

1. For the EPC e of the tainted case of meat, use ONS together with the sequence registry
   (dynamic ONS) to obtain a list of addresses of custodians who have handled the object
   => Set of addresses of custodians' EPCIS services {a}
   May also provide alternative EPCs to track within certain time ranges (e.g. super-tag of pallet)

2. For each address a of an EPCIS service of a party on that supply chain,
   request a trace of the path taken by EPC e while in their custody / premises
   SELECT * FROM reader_data WHERE tagEPC = e
   => Hash-table {(r,t)} consisting of ReaderIDs and corresponding timestamps

3. For each reader r within the trace, request all other EPCs read within a time range t-dt to t+dt
   SELECT * FROM reader_data WHERE readerID = r AND timestamp > (t-delta) AND timestamp < (t+delta) AND tagEPC != e
   => Recordset of observations {(e', r, t')}

4. Iterate steps 2 and 3 over all custodian EPCIS services, a
   => Hash-table of custodians and set of EPCs read at similar times, {(a,{e'})}

5. For each EPC e', perform an ONS lookup to locate the manufacturer's EPCIS service
   => Hash-table of addresses of custodians' EPCIS services {(e',a')}

6. Query the EPCIS of each manufacturer, a' for the description/name of EPC e'
   use XPath to query static attribute data for data keyed on e' and node corresponding to the product description
   => Hash-table of product descriptions {(e',description)}

7. Off-line human evaluation of all product descriptions to determine which products may have caused tainting
   => List of potential contaminant EPCs {contaminantEPCs}

8. Re-use data collected at steps 2-4 to contact each custodian EPCIS where contamination may have occurred.
   => Name and contact details for those custodians

7. ACKNOWLEDGEMENTS

We acknowledge fruitful discussions with members of the Auto-ID Software Action Group and the EPCIS working group.

8. REFERENCES

1. **D.L. Brock, "The Electronic Product Code (EPC) – A Naming Scheme for Physical Objects".**
   January 2001, http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-WH-002.pdf

2 **S. Sarma, D.L. Brock & K.Ashton, "The Networked Physical World – Proposals for Engineering The Next Generation of Computing, Commerce & Automatic Identification".**
   October 2000, http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-WH-001.pdf

3. **URI and URL – Uniform Resource Indicator, Uniform Resource Location.**
   http://www.w3.org/Addressing

4. **S. Sarma, "Towards the 5¢ Tag".**
   November 2001, http://www.autoidcenter.org/research/MIT-AUTOID-WH-006.pdf

5. **M. Harrison & D. McFarlane, "Development of a Prototype PML Server for an Auto-ID Enabled Robotic Manufacturing Environment".**
   February 2003, http://www.autoidcenter.org/publishedresearch/CAM-AUTOID-WH010.pdf

6. **M.G. Harrison, H.J. Moran, J.P. Brusey & D.C. McFarlane, "PML Server Developments".**
   February 2003, http://www.autoidcenter.org/publishedresearch/CAM-AUTOID-WH015.pdf

7. **Oat Systems & MIT Auto-ID Center, "The Object Name Service (ONS) – Version 0.5 (Beta)".**
   http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-TM-004.pdf

8. **Structured Query Language (SQL).**
   see also http://www.sql.org

9. **XPath – XML Path Language.**
   http://www.w3.org/TR/xpath

10. **Oat Systems & MIT Auto-ID Center, February 2002, "The Savant – Version 0.1 (Alpha)".**
    http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-TM-003.pdf

11. **UCCNet.**
    http://www.uccnet.org

12. **Transora.**
    http://www.transora.com

13. **WWRE – WorldWide Retail Exchange.**
    http://www.worldwideretailexchange.com

14. **XML – extensible Markup Language.**
    http://www.w3.org/XML/

15. **XML Schema.**
    http://www.w3.org/XML/Schema

16. **For details of XML markup languages for various sectors of industry,**
    see http://web.mit.edu/mecheng/pml/standards.htm

17. **Mass is amount of substance, weight is the force due to gravity experienced by that mass. Technically, weight is measured in Newtons, whereas the quantity expressed in grams or kilograms is the mass – not the weight.**

18. **XQL – XML Query Language.**
http://www.w3.org/TandS/QL/QL98/pp/xql.html

19. **XML::DOM – XML Document Object Model.**
http://www.w3.org/DOM/

20. **Castor data binding project from Exolab.**
http://castor.exolab.org

21. **Java Architecture for XML Binding (JAXB).**
http://java.sun.com/xml/jaxb/

22. **Java Data Objects (JDO).**
http://java.sun.com/products/jdo/

Designed by Foxner. www.foxner.com