Context and Re-Usability of Core Components

v1.04

Core Components Team

10 May 2001

(This document is the non-normative version formatted for printing, July 2001)

Table of Contents

1 Status of this Document

This document specifies an ebXML Technical Report for the eBusiness community.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

www.ebxml.org/specs/ebCNTXT.pdf

Latest version:

www.ebxml.org/specs/ebCNTXT.pdf

2 ebXML Participants

We would like to recognize the following for their significant participation to the development of this document.

Editing team:

Mike Adcock	APACS
Sue Probert	Commerce One
James Whittle	e CentreUK
Gait Boxman	TIE
Thomas Becker	SAP

Team Leader:

Arofan Gregory	Commerce One

Vice Team Leader:

Eduardo Gutentag	SUN Microsystems

Contributors:

Tom Warner

Jim Dick

Rob Jeavons

David Connelly

Arofan Gregory

Martin Bryan

Mike Adcock

Eduardo Gutentag

Matthew Gertner

Polly Jan

Sharon Kadlec

Sally Wang

James Wertner

Todd Freter

Henrik Reiche

Chris Nelson

Martin Roberts

Samantha Rolefes

3 Introduction

3.1 Summary of contents of document

This document describes how business contexts' influence on data structures can be rendered in an explicit, machine-processable form. This is done by establishing a set of classification hierarchies that are used to identify the situations in which a core component will require modification. The classifications that are being recommended are to be found in ebXML TR - Catalogue of Context Drivers Ver 1.04. The methodology for the use of these context drivers is detailed in ebXML TR – Document Assembly and Context Rules Ver 1.04.

The present document MUST be read in conjunction with these documents. The purpose of this document is to give readers sufficient familiarity with the idea of explicit utilization of context drivers to enable them to understand the classifications and methodology as described in those documents.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119.

3.2 Context defined

When a business process is taking place, the context in which it is taking place can be specified by a set of contextual categories and their associated values. For example, if an glue manufacturer is selling   to a shoe manufacturer, the context values might be as follows:

Contextual Category	Value
Process	Procurement
Product Classification	Glue
Region (buyer)	France
Region (seller)	U.S.
Industry (buyer)	Garment
Industry (seller)	Adhesives

The following set of scenarios explain when context may be applied to a specific Core Component:

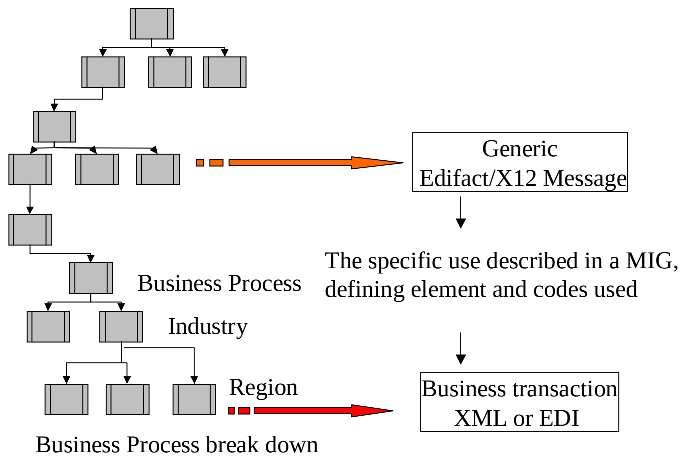
- Design Time - to create the minimum useful schema.

- Integration Time - Identify and help resolve data requirements conflicts required for business transactions.
- Run Time - to express the business relationships between data.
    - Used by Trading Partners to validate the runtime document instances.
- Navigation of the registry to find other data sets.
    - Need to hold the data about the context in the rules.
- Discovery Process for creating Core Components or extensions.
    - Core Components are discovered along with the business context in which they are used.

For a catalogue of Contexts, see ebXML TR - Catalogue of Context Drivers Ver 1.04.

3.3 Context in a business perspective

The concept of context is not new. It can be found already in existing messages like EDIFACT or X12. Context is one of the aspects of modelling business processes, as illustrated in the following example, which shows the top-down modelling of a business process into more and more specific processes:


Generic
Edifact/X12 Message
The specific use described in a MIG, defining element and codes used
Business transaction
XML or EDI
Business Process
Industry
Region
Business Process break down

Although UML modelling of business processes is discussed as if it is a completely new approach, it is not. Earlier development of EDI messages was done by identifying business processes. Typically the underlying process was defined in some generic way, describing the specific data elements and codes to be used, while leaving it to implementations to define the specific use of the message. The Message Implementation Guides describe a subset of a generic message, where specific elements qualified by codes express specific data (semantics). The overall term for this expression of specific data is what we define as *context*.

The diagram above also provides an example of where context is used. Breaking down a business process implies the application of some of the major context drivers.

Context for a business process is one-dimensional, and includes two roles in an industry in a region with respect to an official constraint, for instance. These context drivers are not applied in some sequence: they form *the* context for the business process.

Besides business process context-drivers there may be other activity context-drivers, which again are not applied in some sequence, but form *the* context for the business activity.

The technical application of the Core Components context drivers requires a methodology for using context to define transactions. The business perspective of context is well known and used by implication today. The rest of this technical report, and the ones related to it (ebXML TR - Document Assembly and Context Rules Ver 1.04 and ebXML TR - Catalogue of Context Drivers Ver 1.04) enable an explicit expression and use of context.

4 Using Context Descriptors

4.1 Context-controlled core component metamodel

The formal model for the Context-controlled Core Component Metamodel can be seen in the document ebXML TR - Catalogue of Context Drivers Ver 1.04.

4.1.1 Core component type definitions

A Core Component Type Definition defines a reusable type of core component for which no pre-determined use name has been assigned. No business semantics are associated with the Core Component Type Definition – these semantics appear when it is used in a Basic Information Entity.

Each definition is given a globally unique Identifier, which should be suitable for use as a registry or registry key.

A human-readable name for the type (ending in the word Type, e.g. AmountType), and a brief description of the purpose of the type, are also required. For further specification see the document ebXML TR - CC Dictionary Entry Naming Conventions Ver 1.04.

By default a Core Component Type Definition is deemed to be restrictable or extendable. If this is not the case the isRestrictable or isExtendable boolean properties must be set to False. This is also true of Basic and Aggregate Information Entities.

4.1.2 Basic information entity

Where the types of data that are permitted for a Basic Information Entity are defined by an external agency the name of the maintaining agency and the agency assigned identifier (id) must be recorded.

A formal definition of the relevant Datatype must be associated with each Basic Information Entity. This could be done in accordance with Part 2 of the W3C’s XML Schema specification, or using Document Type Definitions as specified in the W3C XML 1.0 specification.

If a data type is associated with an externally defined list of permitted values, then the URI of a resource that defines the set of currently approved permitted values should be recorded as an external value list object.

If the list of permitted values is defined as part of the core component definition a Permitted Value List must be created. The list consists of one or more Permitted Values identified by a name that is unique within the list, each of which should be assigned one or more Permitted Value Meanings, each of which consists of a statement of the meaning assigned to the value and the IETF RFC1766 language code identifying the language in which the meaning has been defined.

4.1.3 Aggregate information entity

For each component forming part of an Aggregate Information Entity an Aggregation Rules that identifies a Type Use Rules object must be created. The Type Use Rules record the Name assigned to the referenced type within the location and, optionally, an explanation of the use to which the embedded component is being put within this component.

Where there are constraints on the number of times an embedded component can be used these are recorded as the MinMaxConstraints property.

Where there are constraints on the order in which sub-components within the aggregate are to be used an Embedded Group must be defined to identify whether the constraint applies to the use of a choice or sequence of objects.

4.1.4 Functional set

A Functional Set is a set of two or more Functional Sets, or two or more Basic Information Entities or Aggregates that can be used to model information related to a single function in different ways.[1]

4.2 Context constraints

A Document Model is created by applying a set of Context Rules to a set of Basic and/or Aggregate Information Entities that have been “assembled” to meet a defined business process.

The Assemble Types modelling element identifies the base Basic and/or Aggregate Information Entities, applies an appropriate sequence to the components and renames embedded components as required within the business process.

The Context Constraints define modifications to be made to existing Basic and/or Aggregate Information Entities when used within specific contexts, and any Application Component needed to extend a core component or the document model.

1 For example, a location could be recorded as a postal address, a United Nations location code or as a set of co-ordinates as generated by a Global Positioning System. Which of this set of equivalent functions would be chosen for a particular message is context dependent.

Individual constraints are associated with a particular value within a named taxonomy stored as a named context classification within an ebXML repository.

Where the constraint requires that the base definition of a core component be redefined the constraints are defined as a Type Constraint. Where the constraint applies to a facet of a Datatype definition it forms a Datatype Constraint that is associated with a specific Datatype.

4.3 Seeding core components

Lower level core components, either basic or aggregate information entities, can be re-used within higher level aggregates. Fundamentally, they are used "in the context of" the higher level aggregate. This is a purely structural context, not a business context, creating stereotype (i.e. fundamental or generic) information entities.

Recognizing that there are situations in which equivalent information can be expressed in several ways, relevant core components can be grouped together into Functional Sets. These provide a means by which a limited choice of stereotype information entities can be offered as alternative ways of specifying information for a particular function, e.g. a location can be specified as an address, a GPS reference, or a UN Locode. While the functional set is still a stereotype, the choice is dependent on a business context or contexts.

4.4 Using core components

Use of a core component without any modification in a particular business context creates a Substitute Information Entity. This is registered under a unique business name formed from the context and the stereotype component names.

Note This is essential to record the industry sector(s) that use the substitute information entity, the context(s) in which they are used, and all the substitute information entities that use the Core Component.

Use of a core component with extensions (or indeed restrictions) in a particular business context creates a Process Specific Entity. This is registered under a unique business name formed from the context and the stereotype component names.

Note This is essential to record the industry sector(s) that use the substitute information entity, the context(s) in which they are used, and all the process specific entities that use the Core Component.

Substitute information entities and process specific entities are collectively Context Constrained Information Entities. Registration of all these, however numerous, is essential to achieve maximum re-use, to avoid "re-inventing the wheel", and to gain interoperability.

4.5 Building business documents

Business documents are built by drawing on the repository "library" of components. The context descriptors that are registered for each component are used to select the appropriate context constrained information entities for the business document that is being built. These values would be the same as values found in a business process model that informs the contextual use of the core components.

If no appropriate context constrained information entity exists, a new one must be created, according to the principles described in the previous section, and ideally using an existing stereotype. Registration of the new process specific information entity adds to the range of available context-constrained information entities..

4.6 Beyond re-use

If no appropriate existing stereotype exists, an industry vertical or similar community may need to:

- Create additional Basic components for pieces of information, which cannot be represented using already-defined Core Components. These are Domain Basic Components.
- Use Core Component(s) to construct a non-core Aggregate Component, called a Domain Complex Component.
- Use Core Component(s) and Domain Components to construct a non-core Complex Component, also known as a Domain Complex Component.
- Use Domain Component(s) to construct a non-core Complex Component. These are also Domain Complex Components.

Ideally, Domain Components need to be recorded in the same detail as Core Components, complete with relevant Context(s). This is an aspect of extensibility; Domain Components should be registered so as to avoid 're-inventing the wheel'. Newcomers can re-use Domain Components and register any additional Context(s) with which they will henceforth be associated

At some point, non-core Domain Components can become Core Components, according to criteria that judge the degree of re-use. These values would be the same as values found in a business process model that informs the contextual use of the core components.

4.7 Non-compliance issue

This section raises two basic issues:

1. Extensibility

2. Registration

Registering Domain Components cannot be completely policed. Groups or companies might decide to use Core Components, extend them and invent their own Domain Components and never register them.

As a consequence, the use of these Domain Components will not become part of the ebXML standards community. Exact equivalents may well be re-invented in a different way, with different naming, and formally registered as a Domain Components.

Unregistered Domain Components:

- Will hinder communication and interoperability between different communities.
- Should not, in any circumstances, be favoured over formally registered equivalents.

5 The Application of Context to Business Problems

This section offers a discussion of how context can be deployed to solve real-world problems of interoperability and document design. It makes no claims to being comprehensive.

5.1 Promoting interoperability

A few of the common scenarios faced by trading partners today are:

- **Same Data, Different Names:** Frequently, trading partners are asked to support multiple sets of business vocabularies, where the same data is referred to with two or more different names. Typically, the equivalence is established using mapping and translation tools and code conversions, requiring extensive work to integrate systems.

- **Same Data, Different Structural Position:** This is a related problem - the same piece of data may be located in different places structurally in equivalent messages.

- **Same Data, Different Process:** Because of differences in business process, the same data may be expressed differently. Often, this is seen when the same basic message structure is used in two related processes, but the cardinality of some data members is different based on where the message is being used.

- **Same Data, Different Culture:** This is a case most often seen in international trade, where different cultures format and structure data differently from other cultures.

For each of these scenarios, we will look at how the application of context can promote interoperability. In each case, it is assumed that the trading partners describe the data needs for each business process they support in the form of Assembly and Context rules. These can then be made available in a repository, or be given directly to prospective trading partners. Specific implementation options are discussed in more detail below. Please note that all examples given are meant to be illustrative, and may not be based very firmly in reality.

5.1.1 Using context to handle name and structural location variation when determining semantic equivalence

This section addresses the first two scenarios listed above. One place where this type of lack of interoperability is seen is in supply chain scenarios, where small suppliers are selling into more than one industry vertical.

Industry "verticals" are generally defined by the large buyers at the top of the supply chain. Large buyers have highly automated back-office systems; smaller suppliers do not. Because "industries" view things from their own perspective, they tend to organize data differently, and they often use taxonomies that are specific to their industry. Conversely, smaller suppliers often produce goods and services for many different industries: a glue manufacturer could sell a product used in making planes, cars, and shoes, for example, which are seen as three completely separate industry verticals.

Since each industry vertical has different names for the same things, and arranges data differently, it is difficult or impossible for SMEs to fully automate their business. The time for data to travel up and down the supply chain is therefore very long, inventories must be kept high, and many potential efficiencies are lost all throughout the supply chain.

For example, when our small glue manufacturer receives orders from two of these industries, they will have different "standard" vocabularies. Let's say that in the automotive vocabulary, the requested date for shipping each item in an order is called `ShippingDate`, and that this information is always included with each item in the order. For the clothing manufacturer, the same information is a `ShipDate` and it is located only once in the header.

Today, this kind of problem would be handled by translation. A transformation tool would map between these obviously corresponding pieces of data. By analyzing the various vocabularies that must be supported, the glue manufacturer would be able to create a map for each industry standard or trading partner vocabulary supported. The problem here is basically one of cost: an expensive analysis must be conducted to determine the equivalencies in each vocabulary, even when they are fairly obvious.

The automation of this mapping process is enabled by Semantic Identification Documents, which describe a document's derivation from Assembly and Context Rules, and Assembly and Context rules, which describe the derivation of each industry's vocabulary from a set of core components. In each case, the semantics of the data can be identified by tracing them back to the core component from which they were derived.

Because the core component that exists as the basis of any vocabulary can be traced back through this chain, the base semantic of any field or message structure can be determined. By mapping each piece of data in each document structure back to its core, and then comparing the two, equivalence can be automatically determined, and a mapping derived for use by a transformation engine. Note that this process may also require a knowledge of the parent-child relationships between components, as these provide semantic qualification of the core. (For example, a `Tax` element inside a line item has potentially different semantic relevance than the same component used at the header level.)

Ultimately, the cost of developing the mapping for translation technology is reduced, because the extensive manual analysis formerly required is no longer needed. While this does not entirely remove the cost of integrating a new trading partner, it does provide a significant reduction in cost.

5.1.2 Reusing data across related processes

Very often, a single item of data is used in multiple transactions within a single business process, or is used in two related business processes. In many cases, a single message structure can be used to support these different processes or related transactions. An example of this includes an Order, which may be used to request a purchase order (`OrderRequest`), to place an order (`Order`), and to do change ordering (`ChangeOrder`). These three transactions require a nearly identical set of data, but are different. Typically, these differences stem from some action or status related to a specific point in the business process, or involve the ability of one trading partner to include data that may not yet be available at the time a message is created.

In a description of this document structure, fields must be provided for all of the data required at every stage of the process. At the same time, anything that cannot be included at every point across the business process must be made optional. (This is less of an issue with EDI syntax, since all that needs to be changed is the implementation guide that discusses the use of the document. In XML, either an entirely new document type must be described, or a field must be made "optional" that might be better "required" at some other point in the process.)

In order to achieve tight validation, a separate document description for each transaction must be available. If what is wanted is the simplicity offered by having a single document type, then validation must be sacrificed (particularly for XML systems). This is a problem that can be solved through the application of context.

By specifying the needed data and optionality *within a single document type* through context rules, and tying these to a specific transaction or point within the business process, the advantage of smaller, more specific documents, and a single base document type can be achieved. The process described above for tracing a data element back through the Context Rules and Assembly Rules to a specific core component is used again here, although this is typically a design-time activity that does not need to be performed by an application.

5.1.3 International and cultural variation in data

It is very often the case that a single set of business data is structured differently in different parts of the world. Often, this is a reflection of cultural differences in the real world. Perhaps the best-known example of this is the structuring of addresses, which reveal a huge amount of variation. It is certainly possible to store all potentially useful address-related information in a back-office system, but, depending on where the trading partners are and what their data demands are, they will probably only be capable of processing a small number of the possible structural variations.

Context provides a clear way of dealing with this situation: every trading partner can fully describe their structural needs in Context Rules, and the semantic equivalency of different fields can be established using the mechanism described above. This allows us to determine the correct structures for each trading partner, based on where they do business.

5.2 Implementation strategies for core component context

Different use cases will require different implementation strategies for taking advantage of core component context. In the case of smaller companies with minimal back-office software in place, a browser-hosted solution using web forms for data entry may be the best choice for integrating with trading partners. Larger companies will need more sophisticated solutions that bind into ERP systems on the back-end, and provide connectivity with EDI gateways for integration with trading partners who have not implemented ebXML. In both cases, it must be possible to perform integration both at design-time and at run-time. Design-time integration is likely to be the standard case, especially in the short term, but run-time integration will yield the most value over the long-term, since it will enable on-the-fly discovery of new trading partners and negotiation of mutually acceptably data forms, without the need for expensive and time-consuming manual integration work.

5.2.1 Common core component context implementation considerations

In all integration scenarios, the same underlying process is engaged in order to implement core component context. A context engine is fed the appropriate assembly and context rules for both trading partners, identifying the core components that make up the business documents for a given business process and any modifications that must be made to these core components in order to meet specific trading partner requirements.

The assembly rules are applied first, resulting in a schema or DTD modeling the relevant information. (For the sake of simplicity, we will use the term “schema” in subsequent discussion to refer to any one of the various dialects of XML schemas and to DTDs.) Context rules are then applied to adapt the schema to the contexts in which the trading partners are active. The output is thus a customized schema that contains all of the necessary information for the interaction, using standard core components wherever possible to maximize interoperability.

In order to achieve run-time integration, additional information, known as schema annotations, must be made available at the core component level to specify bindings to ERP systems, EDI gateways and web forms. These annotations reference standard core components, once again for interoperability purposes. The annotations, on the other hand, are trading-partner-specific and, in essence, tell the run-time integration engine how to marry these standard core components with the implementation details of the systems used by each company.

5.2.2 Browser-hosted implementation strategy

Small companies that do not have back-office software in place conduct business primarily using phone and fax. For them, manual data processing is an integral part of trading partner integration. Significant value can be gained from use of core components by replacing these existing systems with browser-hosted applications that go straight from a web form to an ebXML-conformant XML document that can be transmitted directly to a trading partner. Conversely, incoming data in the form of XML messages can be displayed in the browser.

If a company wishes to perform design-time integration with a specific trading partner, a schema is first generated that takes into account the requirements of the two parties (using the context engine described above). Two primary interfaces must then be implemented based on the data model described by this schema. The first interface enables the company to view incoming XML documents. This can be achieved by simply applying an XSLT stylesheet to the document to generate an HTML document that can be shown in a browser. The second interface is more complex, and must enable the company to enter data that will be used to create a schema-conformant XML document that will be communicated to the trading partner. This form can be developed using any standard web development technology.

The main advantage of design-time integration is that it does not require any special technology other than what is commonly available today. On the other hand, the manual development of the kinds of sophisticated web forms needed for real-world implementations of complex schemas is quite challenging and time-consuming. The use of tools that automate this process by generating forms directly from schemas can be highly advantageous, to the extent that these tools are available.

In the case of run-time integration, even consultation of incoming documents is more complex than in the design-time scenario. Since the schema is not known ahead of time, so it is not possible to author an XSLT stylesheet to do an XML to HTML mapping. One solution would be to display the documents as raw XML using XML display capabilities such as those included in Internet Explorer 5.0. This is not entirely satisfying, however, as the raw XML view is neither particularly attractive nor intuitive. Otherwise, schema annotations of the type described above can be used to automate the formatting of the document, without the need for a hard-coded stylesheet.

Creation and modification of outgoing documents at run-time clearly requires the use of some sort of tool capable of generating web forms dynamically from schemas. To a large extent, all of the information necessary for this task is contained in the schema itself: structural information, data types, optionality, etc. Additional information such as field labels, length and ordering can be specified using schema annotations. If XML conformant with the input schema is generated when the form is submitted, the result is a full-fledged system for manual interaction in the web browser with ebXML-compliant systems.

5.2.3 ERP and EDI integration

For design-time integration with ERP systems and EDI gateways, the schema generated from the assembly and context rules document is used as the basis for the mapping. One option is to write custom integration code that reads the data from the appropriate system (e.g. BAPI calls to retrieve data from an SAP R/3 database) and generates an XML document that conforms to the schema. This is a fairly straightforward process that can leverage a large body of XML processing software.

Another option is to use one of the increasing number of XML-savvy integration tools. Tools exist already for reading data from a wide range of ERP systems and generating XML documents, and vendors are now announcing support for XML schemas that will partially automate these mappings by reading the schema describing the desired XML document format. The same applies to EDI support; EDI-to-XML gateways exist and are beginning to provide XML schema support that will render the integration task more straightforward.

When run-time integration is a requirement, the same issue arises as with browser-based integration. The schema is not known ahead of time, so it is not possible to write custom code in order to generate XML documents of the appropriate format. The aforementioned schema-aware integration tools for ERP and EDI represent one possible solution to this problem, to the extent that they are capable of fully automating the binding of schemas. As the schema support provided by these systems matures, it is likely that schema annotations of the type described above with also be used to determine which data in the EDI documents or ERP databases corresponds to which data in the generated XML documents.

Clearly integration must work in both directions; i.e. it must be possible to read data from an ERP system, and to write data from an incoming XML document back to the ERP system. In the case of EDI systems it will be necessary to convert from EDI to XML and vice versa. While these cases are not always entirely equivalent (e.g. writing back to an ERP system requires concurrency control that is irrelevant when reading from the system), the differences are implementation details that do not change the overall integration strategy.

6 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

7 Contact Information

Team Leader

Name	Arofan Gregory
Company	Commerce One
Street	Vallco Parkway
City, state, zip/other	Cupertino, CA
Nation	US
Phone:	
Email:	arofan.gregory@commerceone.com

Vice Team Lead

Name	Mike Adcock
Company	APACS
Street	Mercury House, Triton Court, 14 Finsbury Square
City, state, zip/other	London EC2A 1LQ
Nation	UK
Phone:	+44-20-7711-6318
Email:	mike.adcock@apacs.org.uk

Editor

Name	James Whittle
Company	e centre^{UK}
Street	10, Maltravers Street
City, state, zip/other	London WC2R 3BX

Nation	UK
Phone:	+44-20-7655-9022
Email:	james.whittle@e-centre.org.uk