NSRP
Information Technology Panel Project

US Navy Configuration and Logistics Business Data Exchange Specification (DEX)

Development Guide

15 May 2013

Team Members:

Huntington Ingalls Industries - Ingalls Shipbuilding
Northrop Grumman Technical Services
Product Data Services Corporation
Jotne North America
Praeses, LLC

Revision Log

The following is a list of document revisions:

Rev	**Date**	**Reason for Change**	**Page(s)**
0.1	13 September 2012	Initial draft	All
0.2	5 December 2012	Draft for team review	All
0.3	21 January 2013	Draft for team review	All
0.4	28 January 2013	Final Draft for team review	All
1.0	15 February 2013	Final for submittal to NSRP	All
1.1	15 May 2013	Final for publish to NSRP Website	All

TABLE OF CONTENTS

FIGURES

1 Introduction

The DEX Implementation Panel Project was proposed to leverage Product Life Cycle Support (PLCS) work performed on previous National Shipbuilding Research Program (NSRP) and Navy projects to develop a Navy Configuration and Logistics Data Exchange Specification (DEX), implement a prototype data exchange using the DEX, and document the DEX development process and applicable guidance for use on future Navy DEX development projects. The overall goal of the project was to transition previous work on data exchange standards, prototypes, and interoperability into an implementation that can be used on current and future Navy programs. Specific goals included:

- Retain the Navy and Shipyard investment in digital product data throughout the ship life cycle through improved data exchange, migration, and archiving capabilities
- Reduce total ownership costs for future ship classes by developing re-usable DEX reference data for data exchanges and data migrations
- Support future PLCS DEX development and NAVSEA application consolidation efforts by developing a DEX representing the core metadata model for Navy configuration and logistics information
- Facilitate Ship to Shore Connector (SSC) program requirements relative to DEX definition and implementation

This document describes the process and guidance for developing a PLCS Business DEX which can be used for Navy program planning in the areas of data exchange and product model delivery, as well as implementation of PLCS data exchanges on current and future Navy programs. The result of this effort is available from PLCS DEXlib, the PLCS DEX repository[1] (see Reference [1]).

DEXlib versus PLCSlib design decision

Completed publicly available DEXs are kept in a repository. There are also publicly available DEX development environments. It should be noted that the PLCS DEXlib Extensible Markup Language (XML)-based development environment is being migrated to a new System Modeling Language (SysML)-based DEX development environment known as PLCSlib which is still under development. See Reference [4] for the new PLCSlib repository location. The development environments for ISO 10303 AP 239 DEXs are intended to be publicly available[2], and are shared by DEX developers across all industries and projects. This project and future DEX development efforts are dependent upon obtaining access to a stable publicly-available DEX repository and its related software development environment. The existing DEXlib is based on the initial 2005 Edition of ISO 10303-239[3], while the new PLCSlib will be based on a Platform

[1] Navigate to Reference [1]. Under Business Dexs in the navigation pane expand “US Navy”, expand “Business Dexs”, click on “Configuration_and_logistics”.
[2] PLCS DEX development environments are available from
- DEXlib: Reference [3].
- PLCSlib: Reference [5].

[3] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38310

Specific Model (PSM) business object model that is in the process of being harmonized with the upcoming ISO 10303-242[4]. Currently, the new PLCSlib environment is dependent upon purchase of a $2500 commercial SysML software tool, though there are plans to remove this dependency at some point in the future to make PLCSlib free to use by the standards developers. The pros and cons of using DEXlib vs PLCSlib on this project were considered and a decision was made to use DEXlib instead of PLCSlib due mainly to the team's PLCSlib learning curve and lack of maturity of the PLCSlib environment with respect to standard tools and core templates. The team already had DEXlib experience to leverage. This issue will need to be re-evaluated as PLCSlib matures to determine whether it makes sense to transition to the new PLCSlib for future DEX development projects.

[4] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57620

2 Product Life Cycle Support (PLCS) Overview

ISO 10303-239 Product Life Cycle Support (PLCS) specifies the information required to support a product throughout its life. PLCS is one of the ISO 10303 family of standards, generally known as STEP (STandard for the Exchange of Product model data). This family of standards includes the Application Protocols (APs) which define particular business viewpoints of a common information model. The Product Life Cycle Support STEP AP is referred to as ISO 10303-239, AP239, or PLCS.

The scope of a STEP AP is initially described in an Activity Model, using the IDEF0 modeling language, which defines the activities and information flows between activities that the AP can support. The activity model for PLCS defines four major activities:

- Manage information to support a product;
- Generate support solutions;
- Commission support system;
- Provide support.

These major activities are further decomposed and analyzed in the more detailed layers of the Activity Model. The Activity Model serves as the basis of an information model which represents the information produced by or used by the activities described in the Activity Model. It is this information model that provides a formal representation of the data structures that are shared or exchanged during the life cycle of a product. This is illustrated in Figure 1 which shows how a STEP AP defines the scope of the information to be exchanged or shared, and offers a range of exchange formats independent of technology to support that exchange. These formats define the requirements for translators applied to specific systems.

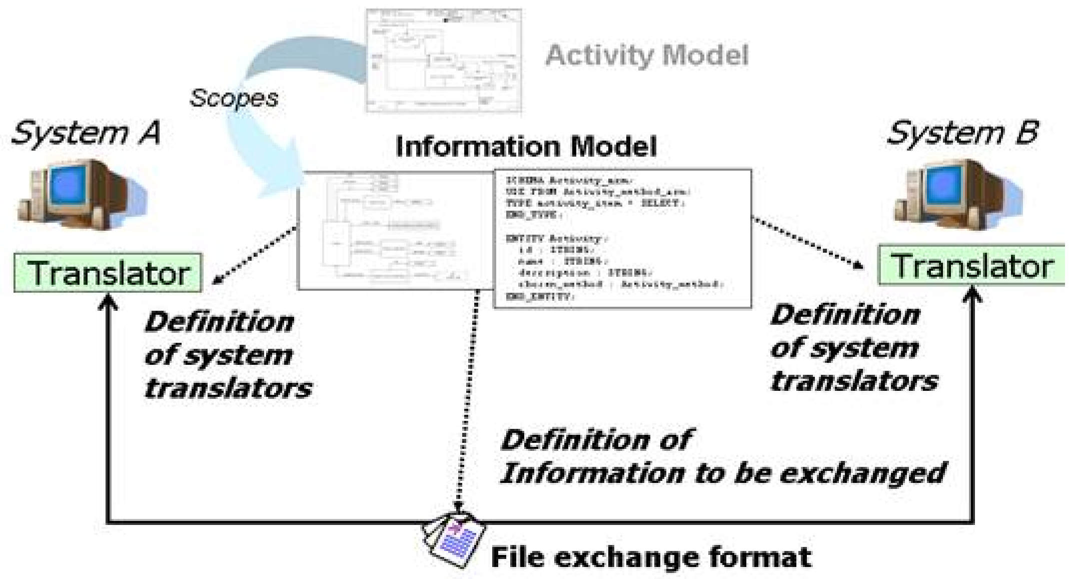

Activity Model
Scopes
System A
Information Model
System B
Translator
Translator
Definition of system translators
Definition of system translators
Definition of Information to be exchanged
File exchange format

Figure 1: Role of a STEP AP

The information model of PLCS is capable of representing all the information required to cover the entire life cycle of a product. It is not, however, aimed at any particular business or industry domain. Because it is designed to be used in many different business applications it is a large, generic information model. While this fact ensures that PLCS has a wide range of application, this also implies that different users will focus on different (though overlapping) areas of the model; and different users, with different information requirements, may interpret the information model in different ways. In order to address these considerations, the PLCS community has developed a number of mechanisms for partitioning the information model into smaller components and for providing additional (and more precise) semantics that add business specific terminology. These include:

- A number of general-purpose subsets of the PLCS model to support the information exchange and sharing requirements of particular activities. These subsets are called Data Exchange Specifications (DEXs). DEXs are built from reusable components (called Templates) that ensure interoperability between DEXs.
- A set of Reference Data that provides greater semantic context to the PLCS model and which may be further tailored to particular business needs. The Reference Data set is stored in a Reference Data Library (RDL).

The need for precise semantics exists at different levels, from the business independent semantics and guidelines provided by PLCS, through business domains (e.g. automotive, aerospace, military), down to specific projects. It is anticipated that the same process (using DEXs, Templates, and Reference Data) is applied at all levels, and that all users of PLCS will take advantage of the starting point given by the work within the Organization for the Advancement of Structured Information Standards (OASIS) PLCS Technical Committee.

For further information on the STEP PLCS standard refer to the PLCS Resources web site (Reference [1]).

3 Data Exchange Specification (DEX) Overview

The information in this section is from the PLCS DEX Information Pages, DEXlib (Reference [1]).

3.1 Data Exchange Specifications (DEXs)

In order to make the PLCS information model suited for many different business contexts, and appropriate for future contexts, it is deliberately generic. Although the PLCS model is generic, it is still quite large. The information model defined by PLCS has a scope that is wider than most applications or any single data exchange. Therefore, it is unlikely that any piece of software will be able to declare compliance to the entire PLCS specification as PLCS covers more capability than most software applications were designed to handle. The large scope also makes it difficult to contract for data to be provided according to PLCS.

The DEX (Data EXchange) Specification concept was developed to address this problem by providing a way of narrowing down the scope of the information model to be used in any given exchange. A DEX is a subset of the PLCS information model, designed to support data exchange for specific activities. DEXs can also be used to contract for information, and software applications may declare conformance to a DEX, thus ensuring interoperability. Figure 2 illustrates a data exchange using a DEX file.

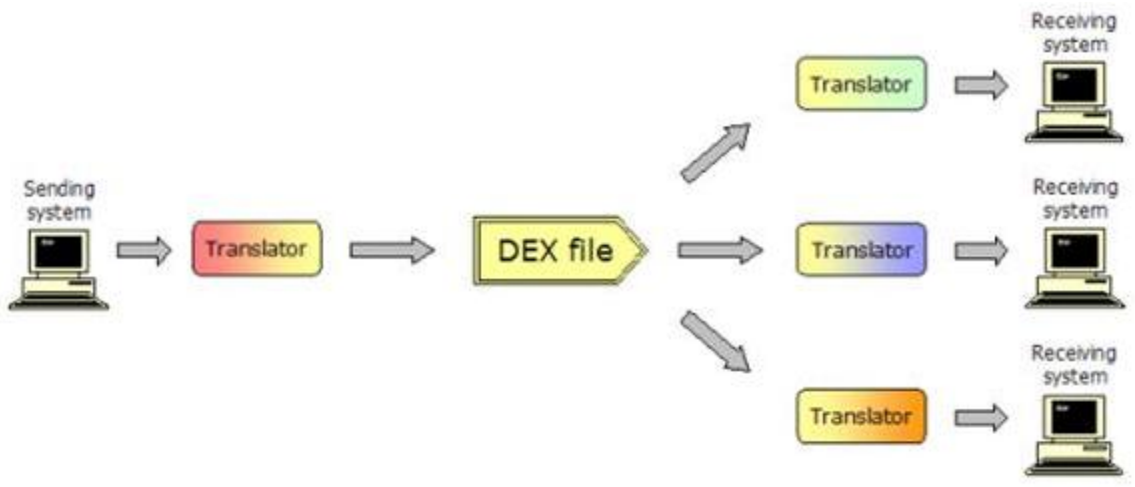


Figure 2: Information Exchange Using a DEX File

DEXs, developed to support common information exchange requirements, are built from Templates. The Templates describe and specify how common business concepts should be represented using the PLCS information model. Templates have been defined at a fine grained level to encourage maximum re-use across different DEXs, hence ensuring consistent usage of the PLCS information model.

As previously discussed, the information model defined by PLCS is generic. It holds no business specific terms. Instead, business semantics are represented by extending the PLCS information model through classification with Reference Data (RD). This provides a mechanism for adapting the generic model to one more specialized for given business domains. The use of the Reference Data is identified within the DEXs and Templates.

In summary:

- A DEX identifies and documents a subset of the PLCS information model required for a specific business purpose;
- The usage of the PLCS information model identified by a DEX is defined through Templates. The Templates provide a precise specification of how the model should be used to represent a given concept;
- Reference Data provides an extensible vocabulary that adds business specific semantics to the information model.

The relationship between these items is illustrated in Figure 3. Further technical details are provided in the following sub-sections.

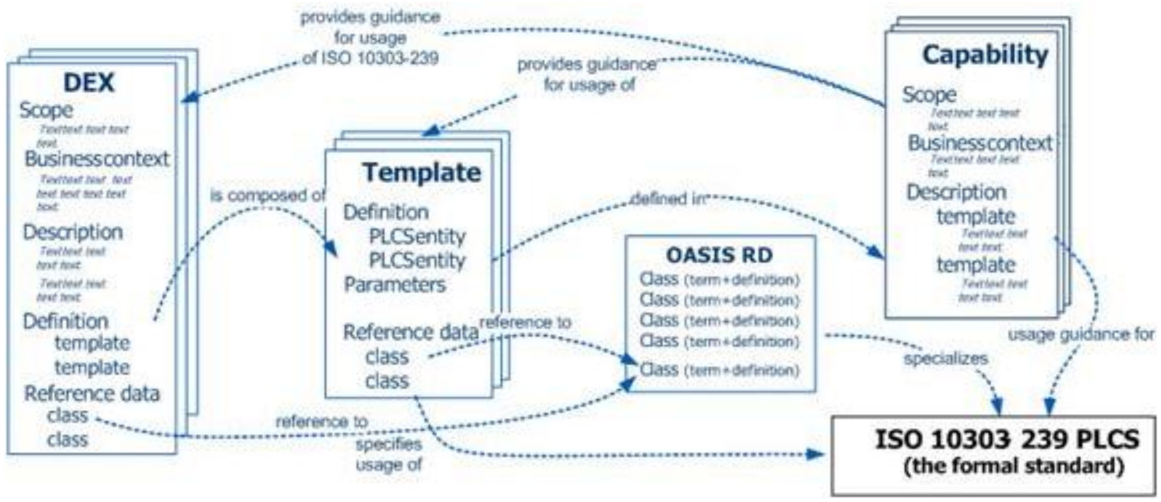


Figure 3: Relationship between DEX Components

A DEX defined and managed by the OASIS PLCS Technical Committee (TC) is referred to as a PLCS DEX. A DEX that has been defined outside the OASIS PLCS TC is referred to as a Business DEX. A Business DEX is, generally speaking, more business specific and may extend or specialize a PLCS DEX. A Business DEX is defined and managed within a specific business context. This Panel Project developed one US Navy Business DEX.

3.2 Templates

A template represents a named abstraction of a grouping of PLCS entities that are to be instantiated in order to represent a concept. A template is analogous to a procedure in procedural programming languages.

A template defines a pattern that identifies precisely which parts of the information model is to be instantiated to represent a given business term. The pattern defined by the template is described both graphically and textually. Each template can then be used in other templates. The template will define:

- What entities to instantiate;
- What attribute values to set on the entities;
- What reference data to be assigned (both mandatory and optional classification).

This Panel Project reused existing templates to create one US Navy Business DEX.

3.3 Reference Data

In order for the PLCS information model to be used in many different business contexts, it is deliberately generic. PLCS entities are provided for representing general constructs such as parts and properties. More precise semantic definition of constructs, such as a safety critical part or the number of gun firings, is not represented directly in the PLCS standard. Instead, provision is made through Reference Data to enable the same precision by classifying the basic constructs and refining or augmenting the meaning of entities.

For example, there are many different types of properties associated with components that are relevant to product life cycle support. Any set of specific property types, such as mean time to failure, that could have been provided by explicit modeling in the standard is likely to be incomplete. Furthermore, as business practices change, different properties are likely to be required over time. The information model allows the properties that can be associated with parts to be classified. Such classification is then used to specify the specific type of property or add a new class of properties.

This approach relies on the use of a common set of classes between partners in a data exchange, together with a shared understanding of what each class means. This is referred to as Reference Data. Reference Data is defined to be life-cycle data that represents information about classes or individuals which are common to many facilities, or of interest to many users.

The Reference Data classes are held in a shared class library referred to as a Reference Data Library (RDL). An RDL is a managed collection of reference data. Reference Data is a key success factor for consistent sharing and integration of data, i.e. to ensure consistent meaning of data. The Reference Data add semantics to the PLCS model.

The standardized PLCS Reference Data is created and published using the W3C Web Ontology Language (OWL). OWL became a W3C Recommendation in February 2004.

There are two types of Reference Data applicable to PLCS DEX usage scenarios:

- PLCS standard Reference Data published through the OASIS PLCS Technical Committee. This is the Reference Data that all PLCS implementations must be able to support and is the basis for the second type of Reference Data.
- Extensions to the standard Reference Data. These may be shared, and perhaps standardized, across an industry sector, or may be project-, contract-, or company-specific.

The standardized PLCS Reference Data is made up of several related datasets as shown in Figure 4:

- a subset of the Dublin Core meta-data elements (plcs-dc) and terms (plcs-dc-terms), with a few OASIS-specific extensions (plcs-express), for internal management of the Reference Data (e.g. creator and date modified);
- the PLCS Standard Reference Data itself (plcs-std-rdl), which may include Classes, a Class hierarchy, Properties of the Classes and Instances or Individuals based on the Classes and relationships that specify which Reference Data elements are applicable for which PLCS schema elements.

Each organization or project may extend the standardized Reference Data (my-org-rdl). Extensions are defined by importing the complete Reference Data Library and adding organization- or project-specific subclasses of the standardized Reference Data Classes. Extensions are intended to be managed in separate datasets and only merged back into the standardized PLCS Reference Data as part of a harmonization, integration, and revision process.

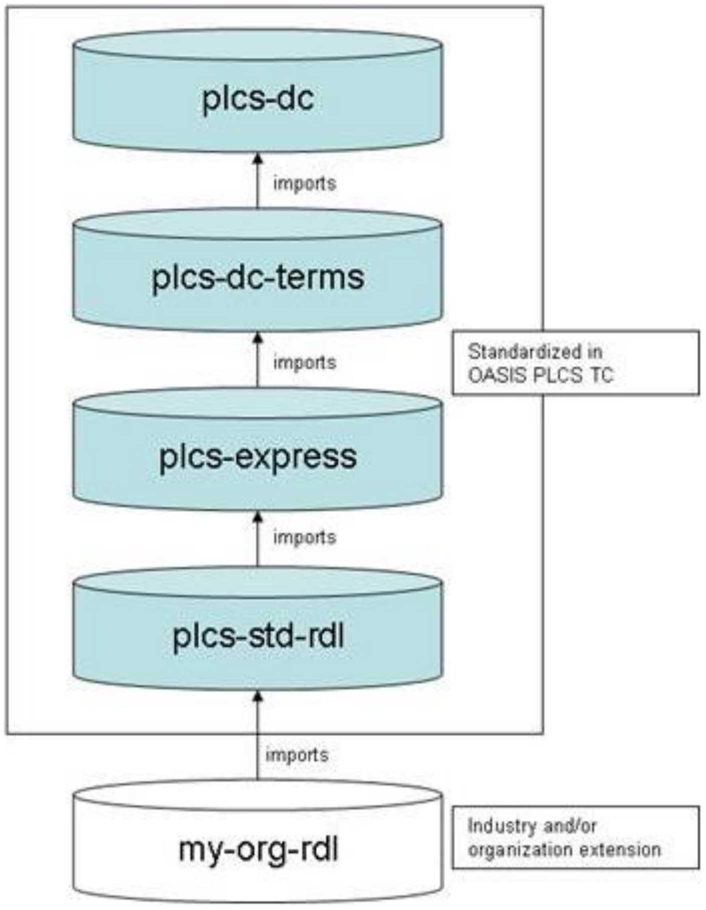


Figure 4: Reference Data Library Architecture

This Panel Project created a new RDL in support of the US Navy Business context.

3.4 DEXlib Repository

DEXlib is the “the repository of information about PLCS, the OASIS PLCS Data EXchange Specifications (DEXs) and other related technology developed by the OASIS PLCS Technical Committee”. It is also used as the repository of Business DEXs and components. A link to existing PLCS and Business DEXs in the DEXlib PLCS Repository is provided in Reference [1].

3.5 DEXlib Development Environment

The DEXlib development environment is “a collection of XML based resources that provide an environment for the definition and development of product data exchange sets for STEP, a family of product data exchange standards, and PLCS, a family of standards related to supporting the complete product life cycle”. These XML based resources are downloaded from Reference [3]. There are numerous free tools that can be used in conjunction with the downloaded resources to speed up DEX development. There are also commercial toolkits that can be used to validate the DEX and instance populations.

3.6 Test Case Development Environment

Once a business DEX has been developed, it must be tested and translators that convert source and target exchange data to/from DEX files conforming to the Business DEX must be developed to support the information exchange represented in Figure 2. Instead of developing full translators, this project developed a prototype test case. The prototype test case included the following products:

- A DEX XML test case file representing the DEX file translated from a notional sending system. This file was created using a partially manual process facilitated by some automated processes using the Jotne Express Data Manager (EDM) tool.
- A set of input files for the receiving system, which were created by translating the DEX XML file into Configuration Data Managers Database-Open Architecture (CDMD-OA) Standard Data Interface File (SDIF) files (see Reference [3]).

The prototype test case development is described later in this document.

4 DEX Development Process

This section describes the high level steps required to develop a business DEX. The intent of this section is to outline and describe the basic steps required to develop a DEX in order for managers and developers to understand the scope of the DEX development process. Additional information for DEX developers and software implementers can be found on the PLCS resources web site provided in Reference [2] and the handbook developed by Jotne (Reference [6]). Although the PLCS resources documentation provides a starting point for DEX developers and software implementers, it lacks detail in many areas and is incomplete in some areas. Therefore, the PLCS resource web site is probably not sufficient to facilitate DEX development by someone unfamiliar with the process, unless it is used in conjunction with advice or support from a company or person who is familiar with DEX development and has experience developing DEXs.

The steps identified in Figure 5 were followed in developing the Navy Configuration and Logistics DEX and are recommended for future Navy DEX development. These steps are described in more detail in the following sub-sections.

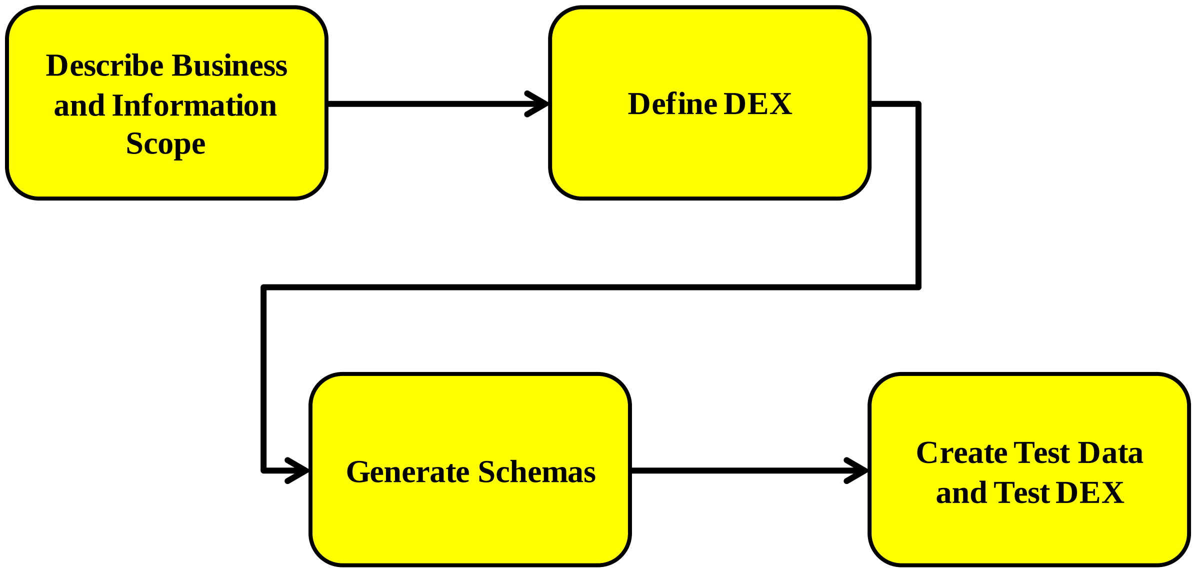

Describe Business and Information Scope
Define DEX
Generate Schemas
Create Test Data and Test DEX

Figure 5: DEX Development Process

4.1 Describe Business and Information Scope

The process for describing the business and information scope can be broken into the activities shown in Figure 6: Describe the Business Scope, Describe the Business Process, and Describe the Business Information Requirements.

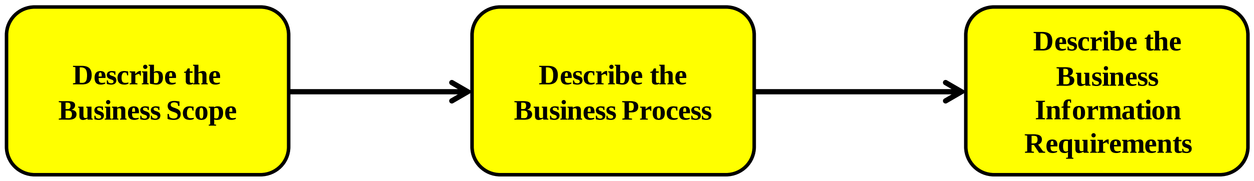

Describe the Business Scope
Describe the Business Process
Describe the Business Information Requirements

Figure 6: Describe Business and Information Scope

The first step in describing the business and information scope of the DEX is to identify the business need for the data exchange. This is an important step because it bounds the problem to a specific business process or processes and determines the scope and context for the data exchange. Identifying the specific business need focuses the problem and sets the stage for developing the data model and DEX in later steps.

Once the specific business need is identified, an investigation should be conducted to determine whether the business need is covered, or partially covered, by an existing DEX. At this point a decision must be made to use an existing DEX, modify an existing DEX, or develop a new DEX. Assuming that a decision is made to develop a new DEX, the next step is to write the DEX introduction and scope statements. This will define the purpose and scope of the DEX and its intended use.

After the DEX has been scoped, the business process in which the DEX is used should be described. The business process will describe key users, activities, systems, and information required to execute the business process. The business process will also include a description of the data exchange scenario which is driving the need for a DEX.

From the business process model, a detailed domain or business data model for the exchange can be developed. This data model will contain the entities, attributes, and relationships among entities that are required for a successful data exchange. The details of the data model define the information requirements for the data exchange and will be used as input requirements for the DEX definition.

This Panel Project defined the business scope as a CDMD-OA data exchange as defined in the Ship Common Information Model (SCIM) (see Reference [9]). Information from the PLCS chapter of the SCIM was used to populate the Configuration_and_logistics Business Information Overview section on DEXlib.

4.2 Define DEX

The process for defining the DEX can be broken into the activities shown in Figure 7: Map Business Data Model to PLCS Data Model, Develop Templates, and Develop Reference Data.

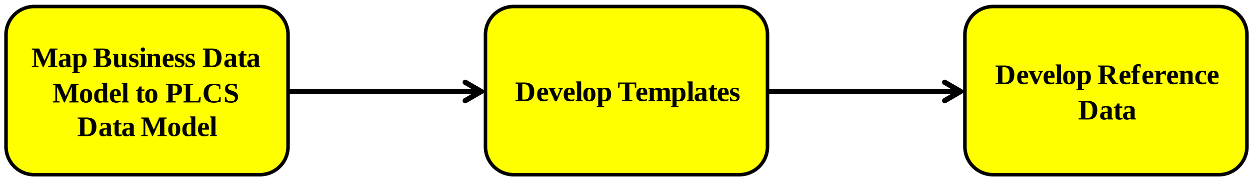


Figure 7: Define DEX

The first step in defining the DEX is to develop a mapping between the domain or business data model and the PLCS entities. This will establish the subset of the PLCS data model that can be used to represent to domain-specific information and facilitate the review of existing PLCS standard templates in the next step.

Once the mapping to PLCS entities is complete, Templates to implement the domain entities need to be identified and developed if necessary as depicted in Figure 8. In some cases, standard OASIS templates and/or business templates exist and can be used to define the DEX. In cases where standard OASIS templates and/or business templates that meet the requirements do not exist, new templates need to be developed.

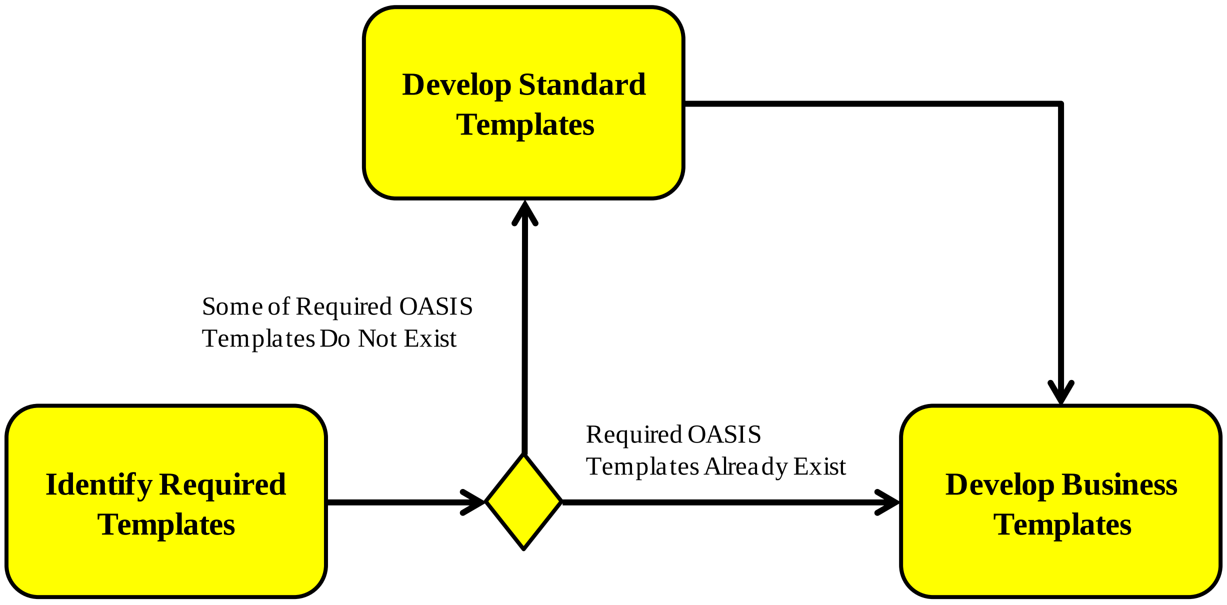

Develop Standard Templates
Some of Required OASIS Templates Do Not Exist
Identify Required Templates
Required OASIS Templates Already Exist
Develop Business Templates

Figure 8: Develop Templates

Templates are described in more detail in the Jotne Handbook (Reference [6]). The Handbook describes the 4 usages of templates outlined below:

1. Template definition - templates are defined in DEXlib where formal parameters are bound to the templates.
2. Template instances in DEXs - template instances are used in various DEX specifications. Some actual parameter values, like class names (reference data)

and relationships to other templates and entities are bound to template instances at this stage.

3. Template instances as mapping targets in an adapter - source schema entity attribute names are mapped to template instance parameters at this stage. This is information related to a specific adapter and hence not found in DEXlib.
4. Population of template - at run time, adapters will populate templates with actual parameter values according to the specifications from the three earlier stages.

This Panel Project used the DEXlib development environment in conjunction with the DEX Template and Graphical Express tools (VISIO 2003 plug-ins) and Oxygen XML editor to develop tailored templates from reused DEXlib templates. Template tailoring to map to the SCIM PLCS schema was accomplished using the Template Tables within the DEX.xml file.

After the Templates are developed, Reference Data is developed according to the steps in Figure 9.

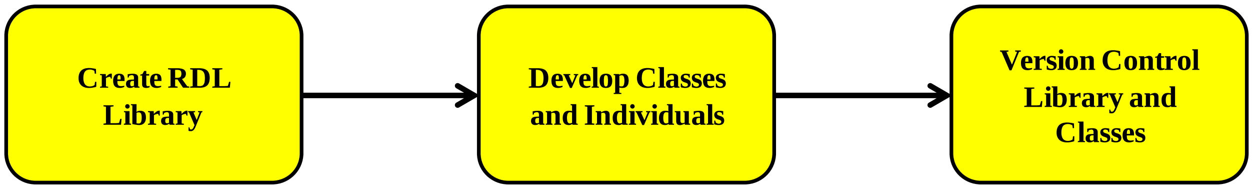

Create RDL Library
Develop Classes and Individuals
Version Control Library and Classes

Figure 9: Develop Reference Data

The Panel Project RDL was created as part of the process of mapping the business requirements to PLCS templates. Use of templates involves the specification of reference data classes. Default classes are provided by the templates and these may be specialized by each project according to the business requirements specified. A RDL is a repository of classes and their relationships. Protégé is a free tool that allows the editing and creation of reference data classes. The RDL is represented using the OWL from the W3C consortium, which is essentially an XML-tagged representation. Standard PLCS template classes are provided within the OASIS RDL identified as “urn:plcs:rdl:std” and a top-level class may appear as “urn:plcs:rdl:std:Part_identification_code”. Each DEX project creates an OWL file for the business context defined that imports the standard reference data library that can then be specialized. DEXLib uses the ‘local’ RDL to verify that the classes used within the DEX.xml exist during the processes of rendering the XML files for html output. Incorrect or missing classes will be marked in red font with a brief error message. In the case of this project, the RDL created for the USN_defense context is identified as ”urn:plcs:rdl:usn_defense” and a specialized class may appear as “urn:plcs:rdl:usn_defense:Record_identification_number”. An OWL file for storing these is defined when creating the business context. The DEX.xml file specifies which templates are to be used and the classes through the AP239_representation section and particularly within the Template Tables. A DEXLib script identifies the classes used in the tables and creates a local RDL.xml file which automatically populates the DEX Reference Data Summary section when the DEX.xml file is rendered in html.

4.3 Generate Schemas

The tools on DEXlib allow for the creation of an Express schema from the documented DEX. The Express schema is then converted to an XML schema and XSD. Note that minor edits to the XML schema were required to enable compilation within common XML tools such as Oxygen and XML Spy.

After the templates and reference data are identified and specified in DEXlib, a utility software tool known as DexPro is used to identify all templates and reference data used and referenced by the DEX. Missing templates are identified and need to be added to the DEX document in DEXlib. With the complete list of required templates in the DEX, the tool then pulls out all the required Express entities from the underlying PLCS schema and builds a temporary new Express schema. The Express schema is checked for referential integrity and errors are reported if any undefined references are found. If errors are found it is usually the result of a missing template or empty select type that requires manual input to resolve. The resolution of these issues is either to regenerate the list of templates or to determine which select type items need to be added to the DEX (there are optional sections to provide these and an xml tool is useful but not necessary for this).

Assuming the Express schema reports no errors, DexPro is then used to generate an XML representation of the Express schema and finally an XSD version. The XSD schema that is generated is not fully complete and requires manual intervention to resolve inconsistencies for acceptable parsing by the two main XML editors (Oxygen and XML Spy). One of these XML editor tools is necessary to identify and fix these issues.

See http://www.plcs-resources.org/plcs/dexlib/help/dex/sw_dexman.htm for more info and instructions for DexPro[5].

4.4 Create Test Data and Test DEX

For this project we developed a populated DEX test case file of sample ship data. This test case file was used to check the initial DEX specification, as published in DEXLib, for correctness and completeness, and used in the DEX to CDMD-OA prototype translator development phase to help the translator developers better understand the intended content of the DEX and the required structure and functionality of the translator code.

As discussed in the previous section, one of the tasks in DEX development is the construction of an EXPRESS schema containing those portions of the PLCS data model that are chosen to fulfill the business requirements for each particular DEX. This EXPRESS schema is a subset of the overall PLCS data model, containing only those PLCS data model entities, attributes, and relationships that are to be populated for data exchanges conforming to the Navy Configuration and Logistics DEX. As the PLCS data model is very generic, most of its business object identification is provided by reference-data class values (contained in Reference Data Library), rather than in the underlying EXPRESS model. The EXPRESS model for the DEX is fairly

[5] Click on Help/Information. Then navigate to Help TOC > Instructions for DEX developers > Developing Business DEXs. Also, navigate to Help TOC > Instructions for DEX developers > Software > DexPro.

compact but each EXPRESS construct or pattern of constructs that make up a DEX Template is reused many times in a data population.

Figure 10 illustrates in Express-G[6] notation the DEX template structure that is used to contain data values for the identifiers of a single part and one of its versions. The part identifier, the organization assigning the part identifier, and the part version information are applied to the generic Part and Part_version entities using the assigning_identification template. The text in blue indicates the data values that are to be applied to the generic PLCS Part entities (Part and Part_version), and what data values are assigned by each application of the assigning_identification template.

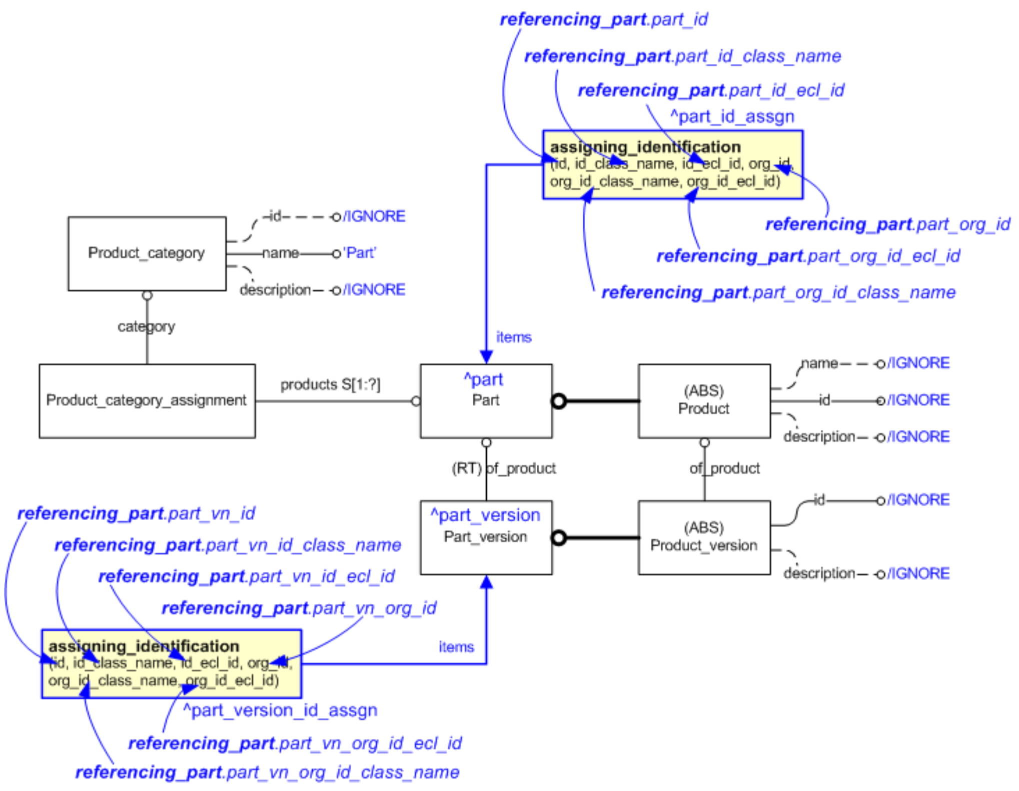

referencing_part.part_id
referencing_part.part_id_class_name
referencing_part.part_id_ecl_id
^part_id_assgn
assigning_identification
(id, id_class_name, id_ecl_id, org_id, org_id_class_name, org_id_ecl_id)
referencing_part.part_org_id
referencing_part.part_org_id_ecl_id
referencing_part.part_org_id_class_name
Product_category
id
/IGNORE
name
'Part'
description
/IGNORE
category
Product_category_assignment
products S[1:?]
items
^part
Part
(ABS)
Product
name
/IGNORE
id
/IGNORE
description
/IGNORE
(RT) of_product
of_product
^part_version
Part_version
(ABS)
Product_version
id
/IGNORE
description
/IGNORE
referencing_part.part_vn_id
referencing_part.part_vn_id_class_name
referencing_part.part_vn_id_ecl_id
referencing_part.part_vn_org_id
assigning_identification
(id, id_class_name, id_ecl_id, org_id, org_id_class_name, org_id_ecl_id)
items
^part_version_id_assgn
referencing_part.part_vn_org_id_ecl_id
referencing_part.part_vn_org_id_class_name

Figure 10: Part and Part Version Population

Figure 11 illustrates the next level of detail within the assigning_identification template structure. The identifier data value for the Part is stored in the Identification_assignment attribute "identifier", and two additional templates (assigning_reference_data and assigning_organization) are used to classify the type of identifier and the "owner" of the identifier. For example, the assigning_reference_data portion may identify the class of identifier for a Part, such as its Record Identification Number (or RIN), and the external class library (ecl) that identifies the Reference Data Library which contains the definition of the RIN. In the case of this project, the Reference Data Library created is called the "USN_defense" library.

[6] http://www.steptools.com/support/stdev_docs/devtools/expg-overview.html

In conjunction with this identifier, the assigning_organization template is used to record which organization “owns” the identifier. The PLCS standard Reference Data Library is used by the assigning_organization template to specify whether the organization that “owns” the part identification is identified by its name, or perhaps instead by its CAGE code.

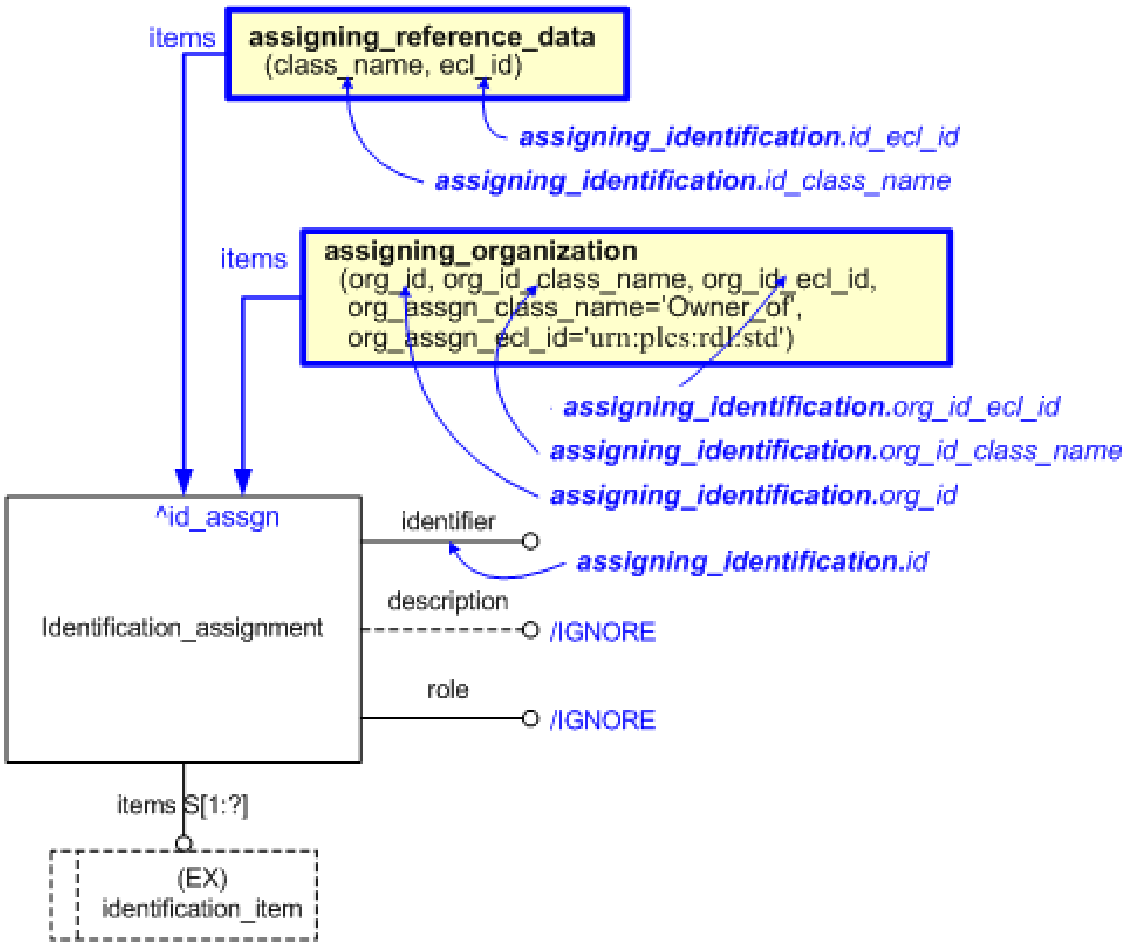

items
assigning_reference_data
(class_name, ecl_id)
assigning_identification.id_ecl_id
assigning_identification.id_class_name
items
assigning_organization
(org_id, org_id_class_name, org_id_ecl_id,
org_assgn_class_name='Owner_of',
org_assgn_ecl_id='urn:plcs:rdl:std')
assigning_identification.org_id_ecl_id
assigning_identification.org_id_class_name
assigning_identification.org_id
^id_assgn
Identification_assignment
identifier
assigning_identification.id
description
/IGNORE
role
/IGNORE
items S[1:?]
(EX)
identification_item

Figure 11: Identification assignment population

A data population of the DEX follows the general guidelines discussed above, using different types of templates to apply various types of data, such as identification, properties, tasks, relationships between parts and the breakdown structures that they are contained in, document information, etc.

The team obtained sample data values for representative parts of a piping system for which configuration breakdown and logistics support record types would normally be maintained for a ship. This data was used to create a realistic test case that complied with the Navy Configuration and Logistics DEX schema and contained the data for population of CDMD-OA Record Types 1, 2, and 3 for several pumps in the piping system.

Jotne's EDMsupervisor toolkit was used to populate the test case file. The DEX EXPRESS schema was imported to create an EDM database conforming to the DEX schema, and the EDMsupervisor user interface was used to create a data population containing instances of the EXPRESS entities, attributes, and relationships in the DEX schema. The user interface allows creation of new instances of each EXPRESS construct in the schema, and population of the data values for the representative piping system parts chosen for the test case. We found that the most efficient method of test case production was to initially populate a representative DEX template structure using the toolkit's user interface for model population, export the partial model to an EXPRESS Part21 file and then copy, paste and manually edit similar template structures in the Part21 file using a text editor. The expanded Part21 file was then re-imported into the EDM database to store the additional data values added during hand-editing. This process of partial population of each new DEX template structure in the EDMsupervisor tool, exporting of the model to Part21 for manual cloning of similar structures, and re-importing of the model to the EDM database was repeated until the model contained all of the required EXPRESS structures and populated ship data for an initial breakdown structure and the CDMD-OA Type 1, 2, and 3 Record Types for a single part. The test case file was later expanded to include additional ship piping data with multiple parts in the breakdown and multiple CDMD-OA Type 3 Records for each part, to accurately reflect the configuration modification and versioning that occurs within the CDMD-OA system over time.

Upon completion of the data population, the test case was exported from the EDMsupervisor tool in both ISO STEP Part 28 (XML) and ISO STEP Part 21 (ASCII) formats and provided to the team for review and for use in development of the DEX to CDMD-OA prototype translator. Although the CDMD-OA translator will ultimately use the DEX XML file as input, we found that the Part21 format is very useful for navigating the data and understanding the structural relationships between the EXPRESS constructs in the schema.

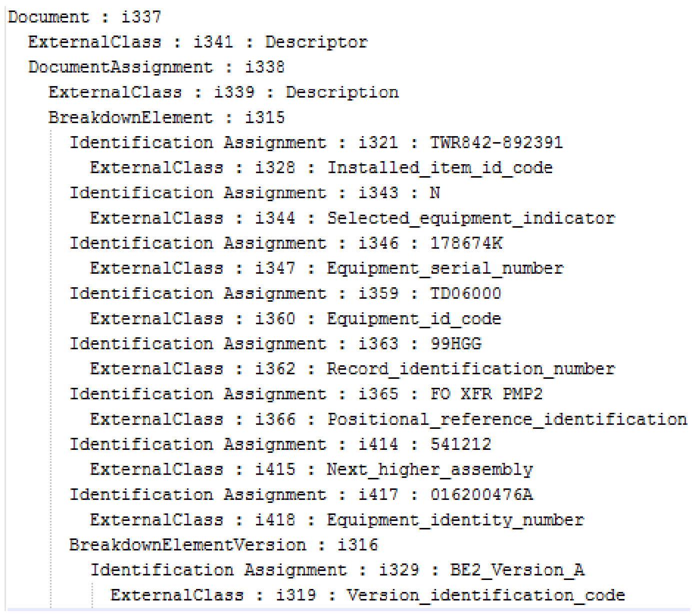
The final test case file in DEX XML file format was then used to translate into a Standard Data Interface File (SDIF) for import into CDMD-OA. The SDIF was imported into CDMD-OA and the CDMD-OA records were reviewed for completeness and accuracy.

The approach used in developing the DEX to CDMD-OA translator (mapping) application was broken into three primary steps:

- Parse the source DEX XML file data into a set of classes (BreakdownElement, Document, etc.)
- Link some of these classes together (LinkBreakdownElement, LinkDocument, etc.)
- Generate the CDMD-OA output (RT2 and RT3 SDIF) from the linked data

A good portion of the effort was spent crafting code addressing the first two steps outlined above. The parser code works very similar to any standard language compiler (compile, link, code gen). The linking process is where the relationships were instantiated in the underlying source data so that the required SDIF files could be built. Once those relationships were in place, the final piece of the application simply pulled out the available RT2 data for the test record and created the SDIF based on the new 750-character length RT2 SDIF definition. The

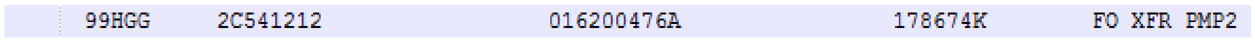
RT2 SDIF is a text file with data values positionally identified within the record. A portion of the interim file created by the parsing and linking processes is shown in Figure 12 below:



```
Document : i337
  ExternalClass : i341 : Descriptor
  DocumentAssignment : i338
    ExternalClass : i339 : Description
    BreakdownElement : i315
      Identification Assignment : i321 : TWR842-892391
        ExternalClass : i328 : Installed_item_id_code
      Identification Assignment : i343 : N
        ExternalClass : i344 : Selected_equipment_indicator
      Identification Assignment : i346 : 178674K
        ExternalClass : i347 : Equipment_serial_number
      Identification Assignment : i359 : TD06000
        ExternalClass : i360 : Equipment_id_code
      Identification Assignment : i363 : 99HGG
        ExternalClass : i362 : Record_identification_number
      Identification Assignment : i365 : FO XFR PMP2
        ExternalClass : i366 : Positional_reference_identification
      Identification Assignment : i414 : 541212
        ExternalClass : i415 : Next_higher_assembly
      Identification Assignment : i417 : 016200476A
        ExternalClass : i418 : Equipment_identity_number
      BreakdownElementVersion : i316
        Identification Assignment : i329 : BE2_Version_A
          ExternalClass : i319 : Version_identification_code
```

Figure 12: Portion of the parsed output from source DEX XML file

The data elements from the source DEX XML file were thus linked (e.g., Record_identification_number = 99HGG) and from that basis the application generated the proper SDIF file. A portion of the RT2 SDIF is shown in Figure 13 with the RIN (99HGG) appearing in position 7 (length of 5 characters) within the file.



```
 99HGG     2C541212                 016200476A              178674K         FO XFR PMP2
```

Figure 13: Part of RT2 SDIF showing RIN (99HGG) beginning in position 7

For RT3, the team was not able to link all the source data elements, thus causing problems in building the correct RT3 file. This is probably related to a linking issue in this version of the code. It should be a minor effort to provide a fix for this issue in future versions of the code moving beyond the current proof-of-concept state into a robust, full-feature version.

This Panel Project used Visual Studio 2012 (C#) and .NET 4.5 to develop the prototype translator.

5 Project Guidance

As mentioned in the Introduction to this Guide, one of the goals of this project was to provide applicable guidance for use on future Navy Business DEX implementations. The following are specific recommendations based on the experience gained during this project that can be used by the Shipyards and Navy to support future projects related to Business DEX development.

- **Business and Information Model development**: For most Shipyard and Navy applications, it is recommended that the Ship Common Information Model (SCIM) (see Reference [9]) be used as the starting point for information model development. Using the SCIM as the basis for data exchange information model development will save time in future DEX development since much of the data model has already been developed. It will also provide a common vocabulary and set of data for DEX development, which will potentially lead to more reuse of templates and reference data.
- **Tools**: Given the complexity of DEX development, the use of software tools is essential to satisfactory completion of the project. Tools are used with DEXlib in the DEX definition phase as well as in the translator development phase. DEX definition is an iterative process using a collection of tools such as Protégé (Reference Data), Visio and plug-ins for Graphical Express & Graphical Template (for Templates) and Oxygen for the DEX itself. The DEXlib development environment is described on the PLCS resources web site (http://www.plcs-resources.org/plcs/dexlib/dex_index.htm).[7] Although we did not implement production translators to/from the DEX XML file format, we did use Jotne's EDMsupervisor toolkit to develop the test case file. Based on the experience with the DEX XML schema gained in this project, it was determined that a tool such as EDM would be essential when developing DEX translators given the complexity of the template instantiation process and use of the reference data in creating a DEX XML instance file.
- **Required Skills/Expertise**: DEX development is not a trivial task, so it is necessary to have support from a contractor with experience with PLCS and DEX development. Our project team had a good mix of domain experts (Navy configuration and logistics subject matter experts with knowledge of CDMD-OA data), translator developers, and PLCS DEX expertise. This combination of expertise worked well in the DEX development process.
- **Cost**: The cost of developing a DEX must include the DEX definition and the translator development. On this project, we developed the DEX, but instead of developing a full implementation with the required translators, we developed a prototype test case to verify the consistency and completeness of the DEX. Depending on the size of the DEX, the complexity of the data relationships, and the ease of mapping to PLCS, the cost could vary substantially from DEX to DEX. For this project we spent approximately $120K to develop a DEX and prototype translator. The portion of the SCIM data model, which we used as a starting point for the DEX information requirements, included approximately 10 entities and 80 attributes. This translated into approximately 334

[7] Click on Help/Information. Then navigate to Help TOC > Instructions for DEX developers > Developing Business DEXs.

templates housing roughly 69 entities, with about 4 attributes for each entity, for a total of 276 attributes in the final DEX definition.

6 Conclusion

This Panel Project was a learning experience relative to the process and technical difficulties involved in developing a Data Exchange Specification. The experience and knowledge gained from this project, and documented in this guide, can be used by the Shipyards and Navy to plan future DEX development activities. A process was documented that leverages existing work performed by NSRP (the SCIM) and a publicly available development environment (DEXlib) to facilitate development of the Navy Configuration and Logistics DEX. In the future, consideration must be made for using the new PLCSlib in place of DEXlib.

The Navy Configuration and Logistics DEX forms the core of the Logistics data developed and maintained for Navy ships and can be expanded to include additional data such as maintenance tasks, provisioning data, training material, and supply support data. DEX development is not a trivial task and requires specific and in-depth knowledge of the PLCS specification, the Express language, and the tools used to develop templates, reference data, and translators.

Members of the Navy Ship-to-Shore Connector (SSC) program team, including Textron and Praeses, participated in this project in order to gain a more in-depth understanding of the DEX development process. The SSC program has a requirement to deliver Product Model data using an AP 239 DEX; however, the program requirement for delivering the Logistics Configuration Baseline is via an MS Access Database, not a DEX. SSC is using the Navy Ship Long-Term Data Retention (ShipLTDR) DEX as the basis for their Product Model data delivery DEX. The ShipLTDR DEX can be found at Reference [1]. The Navy Configuration and Logistics DEX (also found at Reference [1]) is available for the program to use if they are required in the future to deliver program Configuration and Logistics data to the Navy via a DEX.

Project artifacts including the test data, DEX XML files, and translated CDMD-OA SDIF files can be found on the Integrated Shipbuilding Environment (ISE) Tools web site (Reference [11]).

7 Reference Documents

The following documents are referenced or used as source documents for this specification.

No.	Date of Document	Document Number, Title, and Revision
[1]	10 February 2010	DEXlib PLCS Resource web site: http://www.plcs-resources.org/plcs/dexlib/dex_index.htm
[2]	None	DEXlib PLCS DEX Help Table of Contents (TOC): http://www.plcs-resources.org/plcs/dexlib/help/dex/main_toc.htm
[3]	None	DEXlib PLCS development environment: http://dexlib.sourceforge.net
[4]	None	PLCSlib DEX repository: http://www.plcs.org/plcslib/plcslib/
[5]	None	PLCSlib development environment: http://sourceforge.net/projects/plcslib
[6]	12 October 2010	Jotne EPM Technology AS, Handbook for Developing Data Exchange and Sharing Software, using ISO 10303-239, Version 1.3
[7]	7 March 2008	AIA Position Paper: Engineering Data Interoperability - A Standards Based Approach, Version 1.0
[8]	12 February 2009	Aerospace Industry Guidelines for Implementing Interoperability Standards for Engineering Data, Version 1.0
[9]	31 October 2012	Ship Common Information Model (SCIM) Deliverable under Task of NSRP ASE Project Technology Investment Agreement (TIA) #2010-627 http://www.nsrp.org/5-Navy_Product_Data.html http://www.nsrp.org/5-Ad_Hoc/SCIM/SCIM_Docs/SCIM.html
[10]	October 2006	NAVSEA TECHNICAL SPECIFICATION 9090-700D SHIP CONFIGURATION AND LOGISTICS SUPPORT INFORMATION SYSTEM (SCLSIS), PART B, DATA ELEMENT DICTIONARY AND DATA SPECIFICATIONS
[11]	None	Integrated Shipbuilding Environment (ISE) Interoperability Tools http://www.isetools.org/eb-cgi-bin/yabb2_ISE/YaBB.pl

8 Acronyms

The following is a list of acronyms used in this document.

Acronym	Definition
AP	Application Protocol
CAGE	Commercial and Government Entity
CDMD-OA	Configuration Data Managers Database-Open Architecture
EXPRESS	A standard data modeling language for product data. EXPRESS is formalized in the ISO Standard for the Exchange of Product model STEP (ISO 10303), and standardized as ISO 10303-11
DEX	Data Exchange Specification
ISO	International Organization for Standardization
NSRP	National Shipbuilding Research Program
OASIS	Organization for the Advancement of Structured Information Standards (http://www.oasis-open.org)
OWL	Web Ontology Language
PLCS	Product Life Cycle Support
RD	Reference Data
RDL	Reference Data Library
RIN	Record Identification Number
SCIM	Ship Common Information Model
SDIF	Standard Data Interface File
STEP	STandard for the Exchange of Product model data
RT1	CDMD-OA SDIF record type 1
RT2	CDMD-OA SDIF record type 2
RT3	CDMD-OA SDIF record type 3
SysML	Systems Modeling Language
TC	Technical Committee
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition